НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

____ _____ Теплоенергетичний факультет _____

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2019р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 “ Комп’ютерні науки ”

на тему «Web-сервіс генерації гідроакустичного сигналу променевим методом»_____
_____

Виконав (-ла): студент (-ка) 4 курсу, групи ТМ-52

Єлісеєв Михайло Вікторович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник старший викладач Колумбет В.П. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент_____ _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050101 “ Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019р.

ЗАВДАННЯ
на дипломну роботу студенту

Єлісеєв Михайло Вікторович_____
(прізвище, ім’я, по батькові)

1. Тема роботи «Web-сервіс генерації гідроакустичного сигналу променевим методом»_____
_____
керівник роботи Колумбет Вадим Петрович старший викладач_____
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”22”05 2019р. № 1325С

2. Строк подання студентом роботи __17.06.2019-21.06.2019_____

3. Вихідні дані до роботи _платформа Jetbrains PhpStrom 2018.3.3, мова програмування Javascript з фреймворком Angular 7 для клієнтської сторони та мова програмування Node.js для серверної_____
_____

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)__Розробити алгоритм для моделювання гідроакустичного сигналу. Реалізувати код Web-сервісу для цього алгоритму. Створити зручний користувацький інтерфейс для вводу початкових данних та отримання результату у зручному форматі. Візуально зобразити сигнал у вигляді графіка та згенерувати файл dat розширення що буде містити дані цього сигналу для подальшого прослуховування його користувачем._____

5. Перелік ілюстративного матеріалу

«Постановка задачі», «Сфери застосування даного рішення», «Існуючі рішення», «Використані технології», «Архітектура системи», «Головний екран Web-сервісу», «Головні елементи інтерфейсу», «Введення нових направлень», «Алгоритм генерації сигналу», «Кінцевий результат», «Необхідно для користування», «Висновки»._____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”14” жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	09.10.2018	
2.	Вивчення та аналіз задачі	14.10.2019-23.12.2018	
3.	Розробка архітектури та загальної структури системи	02.02.2019-03.03.2019	
4.	Розробка структур окремих підсистем	04.03.2019-14.04.2019	
5.	Програмна реалізація системи	15.04.2019-19.05.2019	
6.	Оформлення пояснювальної записки	20.05.2019-05.06.2019	
7.	Захист програмного продукту	14.05.2019	
8.	Передзахист	28.05.2019	
9.	Захист	17.06.2019-21.06.2019	

Студент _____ Єлсісеєв М.В._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Колумбет В.П._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою роботи було створення веб-сервісу з генерації гідроакустичного сигналу рухомих морських об’єктів променевим методом. Користувацький інтерфейс програми дозволяє ввести вхідні дані, необхідні для розрахунків, серед яких глибина занурення морських об’єктів, початкові та кінцеві координати, їх швидкість, глибина гідрофона, амплітуда хвилі, глибина дна та частота. Після розрахунків сервіс виводить графіки віддаленості морського об’єкта від гідрофона у кожен момент часу, хвильову картину та генерує кінцевий сигнал для будь-якої кількості об’єктів.

Записка містить 52 сторінки, 26 рисунків та 16 посилань.

ABSTRACT

The purpose of the work was to create a web service for generating the hydroacoustic signal of moving marine objects by the radial method. The user interface of the program allows you to enter the input data necessary for the calculation, including the depth of immersion of marine objects, initial and final coordinates, their velocity, depth of the hydrophone, wave amplitude, depth of the bottom and frequency. After calculations, the service displays graphs of the distance of the marine object from the hydrophone at each time point, the wave picture and generates a terminal signal for any number of objects.

The note contains 52 pages, 26 images and 16 references.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

- ГАС — апаратний комплекс, гідроакустична станція, яка використовує явище розповсюдження гідроакустичних хвиль в морях, океанах та інших водних середовищах. Ці станції призначені для вирішення задач навігації, зв’язку, спостереження і т.п.[2]
- FFT — Fast furrier transform
- 3D — 3-dimensional
- ОС — операційна система
- ПЗ — програмне забезпечення
- IDE — Integrated Drive Electronics

ВСТУП

В наші часи гідроакустика знайшла широке використання у безлічі сфер пов’язаних із водним середовищем, а саме підводна локація у будь-яких масштабах, від використання ехолотів для рибалки до локації морських об’єктів у військових діях, зв’язок під водою, виявлення рельєфу дна океанів та морів, пошук затонувших кораблів.

Популярність використання гідроакустичних сигналів для всіх цілей, які були перераховані вище полягає в нестандартній поведінці звуку під водою. В той час як більшість відомих людству методів передачі інформації не є ефективними в морському середовищі, звукові хвилі, через властивість низького затухання, зарекомендували себе як дуже ефективний метод транспортування підводної інформації. Виявилося, що в морському середовищі, звуки поширюються на більші відстані ніж у повітрі.

Для вимірювань звуку у водному середовищі використовується пристрій - гідрофон, який підводним еквівалентом мікрофона. Даний пристрій призначений для запису або прослуховування звуку під водою. Основна маса гідрофонів зроблені на основі п'єзоелектричного перетворювача, який генерує електрику після зміни тиску. Такі матеріали можуть перетворити звуковий сигнал в електричний, так як звук - хвиля тиску.

Головною метою роботи є розробка веб-сервісу, що може генерувати гідроакустичний сигнал морських об'єктів. Цей сервіс є частиною системи, що може отримувати й ідентифікувати сигнали різних водних об’єктів з різних водних середовищ.

Для розробки програмного забезпечення було використано мову програмування Node.js для написання серверної частини і розрахунків, середовище програмування Jetbrains PhpStorm 2018.3, а також мову програмування Javascript з фреймворком Angular для створення користувацького інтерфейсу.

Перший розділ пояснювальної записки містить призначення програмного продукту і постановку задачі; в другому розділі - опис предметної області та

існуючих подібних програмних рішень; третій розділ містить опис системи та засоби реалізації; в четвертому розділі розписано опис реалізації самого програмного продукту; у п’ятому розділі описана робота користувача з програмним продуктом.

1 ПОСТАНОВКА ЗАДАЧІ

Метою дипломної роботи є створення веб-сервісу, що дозволяє генерувати сигнал рухомих морських об’єктів променевим методом.

Променева модель — це метод представлення і розрахування розповсюдження гідроакустичних хвиль. Суть методу полягає в тому, що всі промені являють собою вектора, напрям яких повністю співпадає з вектором нормалі до хвилі. Метод широко використовують при вирішенні прикладних задач[1].

Дана модель здійснює висвітлення обстановки під водою шляхом відстеження траєкторій кожного звукового променю.

Його перевагою є наочність, яка полегшує інтерпретацію гідроакустичного сигналу. Метод широко використовується в сейсмології, оптиці та інших науках за умови, що довжина хвилі набагато менша ніж характерний розмір неоднорідностей середовища, окрім геометричної акустики.

Зазвичай в гідроакустиці використовують хвилі довжиною 7,5-150 см та діапазоні з 1 до 20 кГц.

Постановка задачі дипломної роботи виглядає наступним чином:

1. Розробити алгоритм для моделювання гідроакустичного сигналу, який в якості вхідних даних прийматиме відому кількість відбиттів від поверхні та дна водного середовища. Алгоритм повинен бути реалізований таким чином, щоб на виході можна було отримати фазу на кінці кожного окремого променя хвилі та його довжину у точці ГАС;
2. Реалізувати код програми для моделювання гідроакустичного сигналу;
3. Створити зручний користувацький інтерфейс для генерації сигналу, який включатиме в себе всі необхідні елементи керування для того, щоб користувач мав можливість ввести дані моделювання до системи;
4. Візуально зобразити сигнал та згенерувати його у файл формату dat.

До функціональних можливостей розроблюваної системи були висунуті наступні вимоги:

1. Визначення основних параметрів морського об’єкта;
2. Генерація профілю морського дна;
3. Побудова траєкторії руху морських об’єктів;
4. Генерація гідроакустичного сигналу;
5. Візуалізацію характеристик отриманого сигналу (P, Vx, Vy, Vz);
6. Формування dat файлу з шуканим сигналом.

2 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ ДЛЯ ГЕНЕРАЦІЇ ГІДРОАКУСТИЧНИХ СИГНАЛІВ

Існуючі системи для генерації гідроакустичних сигналів є програмним забезпеченням, що створені для моніторингу та візуалізації результатів роботи різних типів гідрофонів, які використовуються для виконання різноманітних завдань. Розглянемо приклади таких програмних продуктів.

2.1 Програмне забезпечення Lucy Software

Lucy Software — це програмне забезпечення, розроблене для персональних комп'ютерів та ноутбуків , яке дозволяє переглядати та взаємодіяти з даними, які були зібрані з гідрофонів icListen Smart Hydrophones. Цей програмний продукт показує візуалізацію хвиль у реальному часі. Будування графіків за допомогою Lucy дозволяє проводити точні вимірювання з записаних або отриманих у реальному часі даних.[7]

Переваги програмного продукту Lucy (Рисунок 2.1 та рисунок 2.2):

1. Взаємодіють з усіма продуктами IcList Smart Hydrophone
2. Існує можливість контролювати параметри збору даних у гідрофоні
3. Існує можливість отримувати акустичні дані в реальному часі з icListen та переглядати їх на головному екрані сервісу.
4. Одночасна взаємодія з кількома сигналами
5. Отримання більш чіткого сигналу фільтруванням даних з допомогою спеціального інструменту для скасування шуму Lucy.
6. Перегляд бібліотеки даних з використанням інструменту Replay.

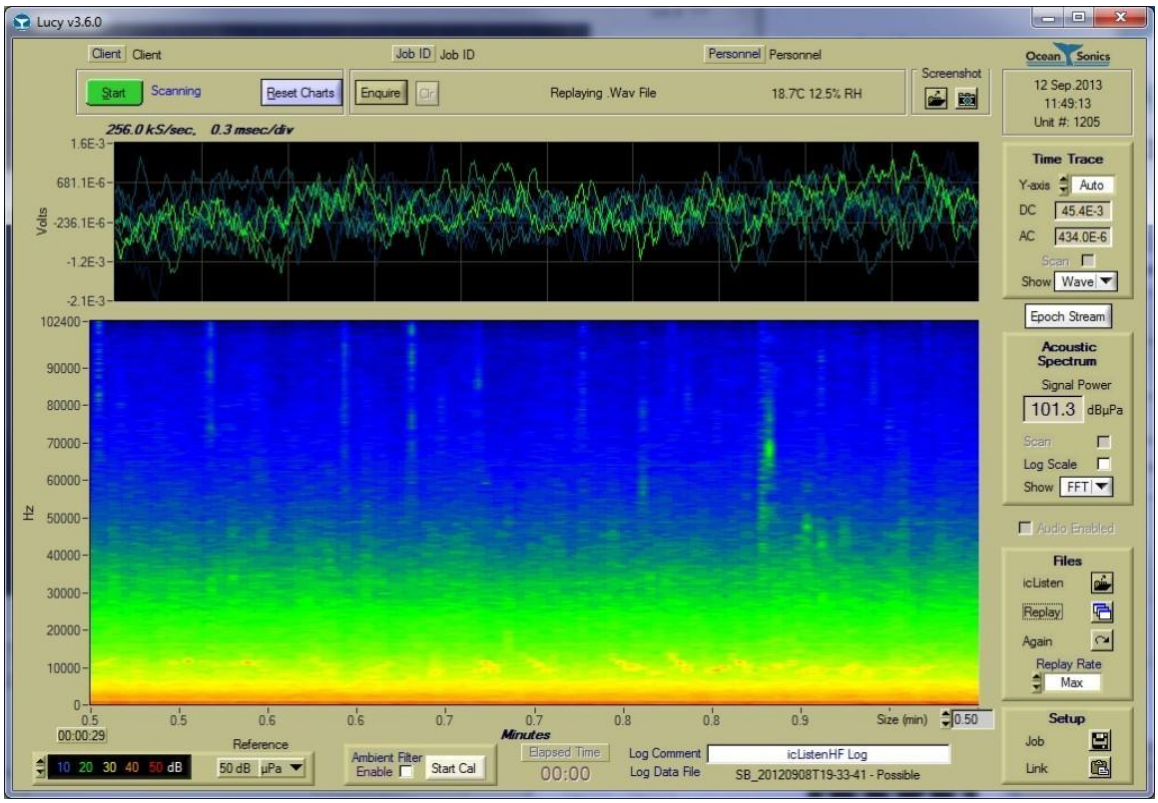

Рисунок 2.1 — Приклад роботи Lucy Software після заміру шуму китів у Тихому океані

Області застосування програми Lucy:

1. Вимірювання шумів у водних середовищах
2. Моніторинг об’єктів під водою
3. Наукові та океанографічні дослідження
4. Моніторинг життєдіяльності морських тварин
5. Виявлення витоків трубопроводів
6. Екологічний моніторинг

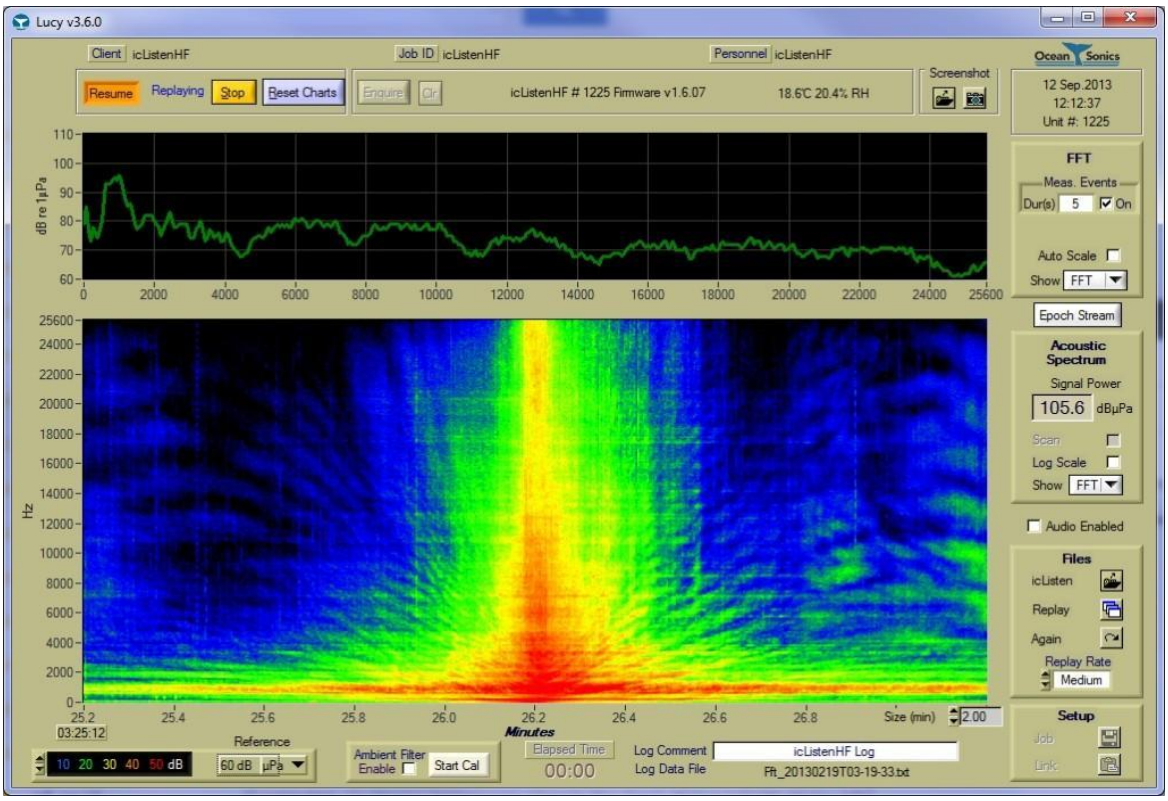

Рисунок 2.2 — Приклад роботи Lucy Software під час виміру шуму судна, що пропливає повз гідрофон

Режими роботи Lucy Software:

1. Режим реального часу
    1) Візуальне відображення акустичних даних, під час виміру;
    2) Демонстрація часових рядів FFT;
    3) Збереження початкових даних;
2. Режим скасування шуму
    1) Можливість переглядати шуми під час моделювання та подальше видалення його з дисплею
    2) Початкові дані залишаються незмінними, не зважаючи на те, чи фільтрація, чин і
3. Режим відтворення шуму
    1) Візуальне відображення зібраних акустичних даних;
    2) Перегляд FFT та даних сигналу.

2.2 Програмний продукт SpectraPLUS-SC

Зразок цього програмного продукту, створено компанією Cetacean Research Technology разом із компанією Pioneer Hill Software, LLC. Програма SpectraPLUS-SC головним чином використовується для спектрального аналізу підводних звуків та у галузі біоакустики. (Рисунок 2.3)

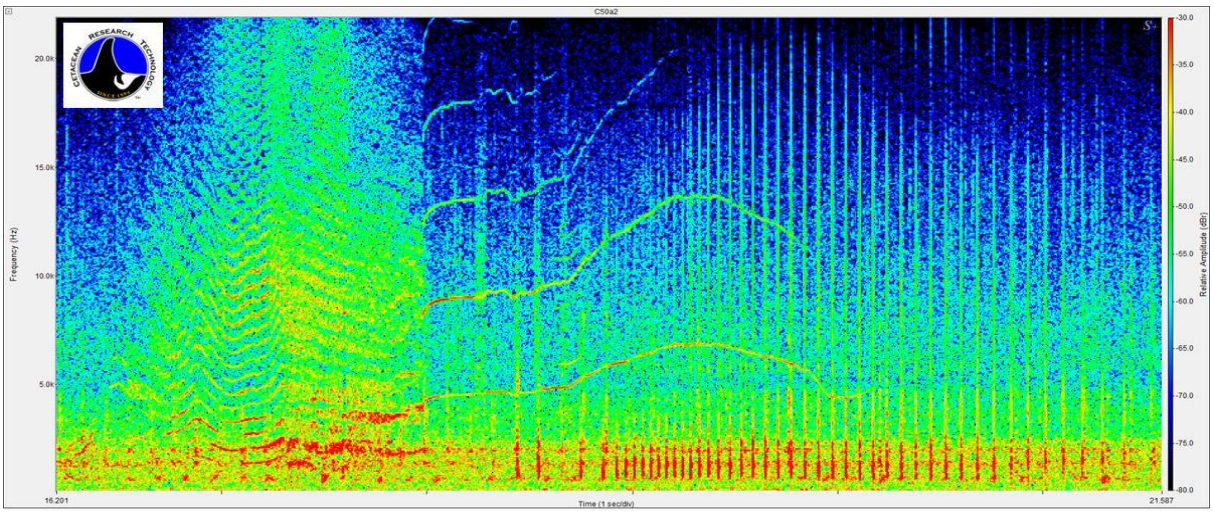

Рисунок 2.3 — Спектрограма

SpectraPLUS-SC забезпечує запис, обробку та відтворення до двох каналів в режимі реального часу. Також, вона має комплексний набір дисплеїв для відображення спектру, фазових і часових вимірювань, та частотного аналізу (3D ділянки поверхні та спектрограма). Доступний широкий вибір утиліт та інструментів для генерації сигналів, реверберації та зменшення шумів, а також можливість автоматизації аналізатора для випробувань на виробництві та інших потреб.

Також цей програмний продукт має безліч інших переваг. Наприклад: генерація сигналів з використанням білого або рожевого шуму, частотний підбір, частотний крок, генерація сигналів у різних каналах, аналіз спотворення сигналу, а ще функція «3D Surface View», яка відображає спектр порівняний з часом у 3D форматі.

2.3 Висновки до розділу

В цьому розділі було розглянуто два програмних продукти, призначені для моніторингу результатів роботи гідрофонів:

1. Lucy Software;
2. SpectraPLUS-SC.

Була визначена область застосування цих програмних продуктів, їх функціональні можливості та різноманітність режимів роботи.

3 ЗАСОБИ РОЗРОБКИ

При розробці програмного продукту важливим є правильний вибір технологій та засобів програмної реалізації, що впливає на якість, час розробки, швидкість роботи продукту та надійність. Основним середовищем розробки було обрано середовище Jetbrains PhpStorm 2018.3.3.

Для розробки саме алгоритмів розрахунків використовувалась мова програмування Node.js.

Для графічного інтерфейсу було використано мову програмування Javascript з фреймворком Angular 7.

3.1 Середовище розробки Jetbrains PhpStorm 2018.3.3

JetBrains PhpStorm — крос-платформне середовище розробки для PHP та інших мов програмування для web-сервісів, яке розробляє компанія JetBrains. (Рисунок 3.1)

PhpStorm є інтелектуальним редактором для PHP, HTML і JavaScript з можливостями аналізу коду до компіляції, запобігання помилок у коді і автоматизованими засобами для рефакторинга PHP і JavaScript коду. Також PhpStorm підтримує автодоповнення коду для швидкого програмування і запобігання опечаток. Воно включає в себе співпрограми, генератори, замикання, типажі, синтаксис коротких масивів і простори імен. Також в наявності SQL-редактор з можливістю переглядати таблиці та робити запити, не виходячи з IDE.

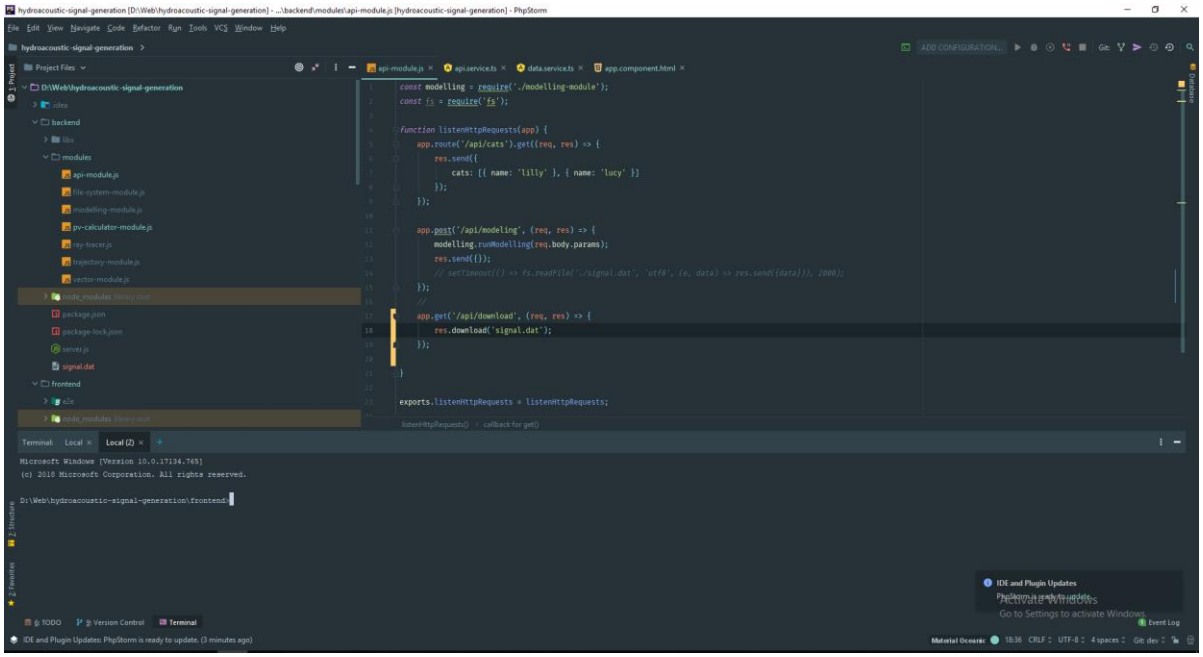

Рисунок 3.1 — Середовище розробки Jetbrains PhpStorm 2018.3.3

PhpStorm був розроблений на основі середовища IntelliJ IDEA, написаної на мові програмування Java, та використовується для розробки та компіляції на Java. Користувачі можуть розширити функціонал середовища розробки за рахунок установки готових плагінів, або написавши власні плагіни для своїх цілей.

Робота із JavaScript, CSS і HTML

- Автодоповнення коду для мов програмування JavaScript, HTML і CSS (для ключових слів, тегів, змінних, параметрів та функцій).
- Підтримка CSS3 і HTML5
- Live Edit: зміни в коді можна відразу переглянути в браузері навіть без перезавантаження веб-сторінки
- Підтримка розмітки стилів CSS/LESS /SCSS/SASS(підсвічування помилок, валідація, автодоповнення коду тощо)
- Пошук використань та ініціалізації змінних, функцій тощо
- Рефакторинг JavaScript (виділення, перейменування, змінної / функції, переміщення / копіювання, вбудовування функції / змінної)
- Інтеграція JavaScript з фреймворками, такими як Angular, React, Vue та інші.

- Робота з системами контролю версій — зручний інтерфейс для додавання змін до віддаленого репозиторію та стягування їх звідти, а також вирішення конфліктів в коді різних версій

3.2 Середовище виконання Node.js

Node.js — мова програмування для виконання продуктивних мережевих застосунків, написаних за допомогою мови JavaScript.[5] Засновником мови є Раян Дал.[14] Раніше Javascript застосовувався для обробки даних в браузері тільки на стороні користувача, а node.js дав можливість виконувати скрипти Javascript на сервері та віддавати користувачеві результат виконання даних. Node.js перетворила JavaScript на мову загального використання та дозволила FrontEnd розробниками писати примітивну серверну частину коду у необхідних випадках, без допомоги BackEnd розробників і без вивчення інших мов програмування

Node.js має наступні властивості:

- однопотокова асинхронна модель виконання запитів;
- неблокуючий ввід та вивід;
- система з модулів CommonJS;
- компілятор JavaScript Google V8;

Для керування модулями використовується менеджер пакетів npm (node package manager).[15]

3.3 Мова програмування Javascript

JavaScript (JS) — об'єктно-орієнтована, прототипна, динамічна мова програмування. Програмна реалізація стандарту ECMAScript. Найчастіше вона використовується для оживлення веб-сторінок, що надає можливість на стороні клієнта — пристрої користувача, мати контроль над браузером, взаємодіяти з користувачем, асинхронно передавати дані на сервер та отримувати їх від нього,

змінювати зовнішній вигляд та структуру веб-сторінки.

JavaScript характеризують як скриптову та прототипну мову програмування з динамічними типами. Також JavaScript частково підтримує й інші парадигми програмування, такі як: функціональну та імперативну і деякі властивості архітектури як: слабка та динамічна типізація, керування пам'яттю автоматично, функції як об'єкти першого класу та прототипне наслідування.[8]

Мова JavaScript використовується для:

- написання сценаріїв (скриптів) сторінок для надання їм інтерактивності та динамічності;
- створення односторінкових веб-застосунків — SPA (Single Page Application) за допомогою різних фреймворків (React, Angular, Vue тощо);
- написання коду на стороні сервера за допомогою платформи Node.js;
- створення десктопних додатків за допомогою технологій Electron чи NW.js;
- мобільних додатків (React Native, Ionic);
- скриптів в прикладних ПЗ (наприклад, в програмних продуктах як Apache JMeter або Adobe Creative Suite);

Мови програмування Java та JavaScript дуже схожі за назвою, але є двома різними мовами. Мова Javascript з’явилась пізніше, ніж Java і така назва була дана для її популяризації, так як Java була поширена в той час. Вони мають різну семантику, хоч і мають схожі риси в правилах іменування та стандартних бібліотеках. Синтаксис обох мов був отриманий від мови C, але дизайн та семантика JavaScript є результатом впливу таких мов як Scheme та Self.

Структурно JavaScript можна представити у вигляді об'єднання трьох чітко помітних одна від одної частин:

- ядро (ECMAScript),
- об’єктна модель браузера (Browser Object Model або BOM),
- об’єктна модель документа (Document Object Model або DOM).

Якщо розглядати JavaScript в відмінних від браузера середовищах, то об'єктна

модель браузера і об'єктна модель документа можуть не підтримуватися.

Об'єктну модель документа іноді розглядають як окрему від JavaScript сутність, що узгоджується з визначенням DOM як незалежного від мови інтерфейсу документа. На противагу цьому ряд авторів знаходить BOM і DOM тісно взаємопов'язаними.

3.3.1 Ядро

ECMAScript не є браузерні мовою і в ньому не визначаються методи введення і виведення інформації. Це, скоріше, основа для побудови скриптових мов. Специфікація ECMAScript описує типи даних, інструкції, ключові і зарезервовані слова, оператори, об'єкти, регулярні вирази, не обмежуючи авторів похідних мов в розширенні їх новими складовими.

3.3.2 Об’єктна модель браузера

Об'єктна модель браузера - браузер-специфічна частина мови, що є прошарком між ядром і об'єктною моделлю документа. Основне призначення об'єктної моделі браузера - управління вікнами браузера і забезпечення їх взаємодії. Кожне з вікон браузера представляється об'єктом window, центральним об'єктом DOM. Об'єктна модель браузера на даний момент не стандартизована, однак специфікація знаходиться в розробці WHATWG і W3C.

Крім управління вікнами, в рамках об'єктної моделі браузера, браузерами зазвичай забезпечується підтримка наступних сутностей:

- управління фреймами,
- підтримка затримки в виконанні коду та зациклюванні з затримкою,
- системні діалоги,
- управління адресої відкритої сторінки,

- управління інформацією про браузер,
- управління інформацією про параметри монітора,
- обмежене управління історією перегляду сторінок,
- підтримка роботи з HTTP cookie.

3.3.3 Об’єктна модель документа

Об'єктна модель документа - інтерфейс програмування додатків для HTML і XML-документів. Згідно DOM, документ (наприклад, веб-сторінка) може бути представлений у вигляді дерева об'єктів, що володіють рядом властивостей, які дозволяють виробляти з ним різні маніпуляції:

- генерація і додавання вузлів,
- отримання вузлів,
- зміна вузлів,
- зміна зв'язків між вузлами,
- видалення вузлів.

3.4 Метамова Sass

Sass — від англійського Syntactically Awesome Stylesheets — метамова із скриптів, що інтерпретується в CSS (каскадні стилі).

Спроектований Sass був Гемптоном Кетліном і розроблений Наталі Вейзенбаум. Він призначений для збільшення абстракції коду, спрощення та зменшення файлів каскадних стилів.

Ця мова має два синтаксиси:

- Sass (original) — відрізняється від CSS відсутністю крапок з комою та фігурних дужок. Щоб зробити вкладений селектор потрібно вставити необхідну кількість відступів

- Scss (new) — більш схожий на CSS та потребує фігурних дужок.

Файли scss-синтаксису мають розширення .scss, в той час як sass-синтаксису — .sass.

Мова Sass розширює CSS та надає можливість механізми, схожі до інших, традиційних мов програмування, зокрема мовах об'єктно-орієнтованого виду та недоступних для каскадних стилів.

Інтерпретатор Sass компілює скрипти Sass у блоки CSS. Sass є синтаксичним цукром для CSS.

3.5 Фреймворк Angular

Angular (фреймворк Angular 2 або Angular 2+, тобто друга та вищі версії) — front-end фреймворк з відкритим кодом написаний на TypeScript, який розробляється та підтримується Angular Team під керівництвом компанії Google і спільнотою корпорацій та приватних розробників. [4] До нового фреймворку Angular у відкритому доступі був фреймворк AngularJS (Angular 1). Вони між собою відрізняються у багатьох аспектах. Angular — це AngularJS, який переосмислили та який був повністю переписаний тією ж командою розробників.[12]

Основними елементами при розробці є компоненти, модулі, шаблони, сервіси, директиви, інтерфейси, моделі та ін'єкції залежностей.[16]

Як було згадано вище, Angular — це ретельно переписаний AngularJS.

- Було додано Angular CLI, що дозволяє створювати нового проекту, та будь-якого елементу (з наведених вище) з правильною структурою і розташуванням в проекті
- Angular не використовує "області видимості" ($scope) та контролери, а застосовує ієрархію компонентів, як головну концепцію архітектури
- Angular має відмінний синтаксис написання атрибутів в шаблоні, застосовуючи "квадратні дужки" для біндінгу динамічних даних доступних

властивостей, і "круглі дужки" для біндінгу динамічних даних подій ($event)

- Значна частина функціоналу Angular перенесена у модулі, що називається Модульність
- Angular застосовує та рекомендує мову — TypeScript, що повторює весь функціонал Javascript та має такі додаткові можливості, як:
    - Класи, а отже підтримує Об'єктно-орієнтоване програмування
    - Система типізації, що дозволяє передбачити помилки при збірці проекту та відслідковувати типи даних які передаються та повертаються в функції або методі
    - Узагальнений тип програмування
- Присутність таких ES6-можливостей, як:
    - Ітератори
    - Анонімні функції
    - Стрілочні функції
    - Цикли типу For/Of
    - Генератори
    - Рефлексія
    - Деструктуризація
- Динамічне завантаження даних
- Асинхронна компіляція HTML шаблонів
- Заміна контролерів та областей видимості ($scope) директивами та компонентами – директивами з шаблоном
- Ітераційні колбеки з використанням RxJS. RxJS обмежує видимість станів і можливість дебагу, але, застосовуючи такий плагіни, як ngrx, це легко вирішується.

3.6 Бібліотека CanvasJS

Canvas — елемент HTML5, який можна застосовувати для малювання графіки

використовуючи скрипти (переважно JavaScript). Наприклад його можна застосувати для малювання графів, створення фотокомпозицій а також анімації.

Елемент <canvas> є частиною специфікації HTML та W3C HTML Canvas 2D Context. <canvas> вперше було втілено Apple в Mac OS X Dashboard та Safari 3.1. У Gecko підтримка <canvas> з'явилася у версії Firefox 1.5, у Presto з версії 9.0 веб-браузера Opera, а Internet Explorer підтримує <canvas> починаючи з 9-ї версії.

Щоб відобразити <canvas> в html-документі, слід використати наступний код:

```
<canvas id="tutorial" width="150" height="150"></canvas>
```

Він дуже схожий на тег <img>, з тією лише різницею, що не містить атрибутів src і alt. Елемент <canvas> має всього два атрибути — width і height. Обидва вони не є обов'язковими, і можуть бути задані через властивості DOM. Якщо ширина і висота не визначені, <canvas> буде створений шириною в 300 пікселів і 150 пікселів заввишки. Розмір елемента може бути довільним і задаватися через CSS, але при промальовуванні картинка масштабується відповідно до компонування.

Атрибут id не є специфічним для елемента <canvas>, але є одним з атрибутів HTML за замовчуванням, і може бути застосований майже до всіх елементів HTML (також як class, наприклад). Завжди визначати id елемента — гарна ідея, тому що це значно спрощує ідентифікацію його за допомогою скриптів.

Стиль елемента <canvas> може налаштовуватися також, як і звичайне зображення через CSS (margin, border, background, і т.п.). Ці правила, проте, не впливають на саме малювання в <canvas>. Якщо ніякі налаштування стилю не задані, <canvas> буде створений повністю прозорим.

Canvas створює поверхню для малювання, яка надає один або більше контекстів для відтворення, який використовується для створення відображуваного контенту і маніпуляцій з ним. Ми сфокусуємо на 2D (двомірному) контексті відтворення, який в наш час є єдиним певним контекстом. У майбутньому інші

контексти зможуть підтримувати інші види відтворення: наприклад, цілком ймовірно, що буде додано 3D контекст, заснований на OpenGL ES.

<canvas> спочатку порожній, і для того, щоб що-небудь відобразити, скрипту необхідно отримати контекст відтворення і малювати вже на ньому. Елемент <canvas> має DOM-метод getContext і призначений для отримання контексту відтворення разом з його функціями малювання. getContext () приймає один параметр — тип контексту

3.7 Стандарт мови програмування ECMAScript

ECMAScript — стандарт мови програмування, затверджений міжнародною організацією ECMA згідно зі специфікацією ECMA-262. Найвідомішими реалізаціями стандарту є мови JavaScript, JScript та ActionScript, які широко використовується у Вебі.

JavaScript має низку властивостей об'єктно-орієнтованої мови, але завдяки концепції прототипів підтримка об'єктів в ньому відрізняється від традиційних мов ООП. Крім того, JavaScript має ряд властивостей, притаманних функціональним мовам, — функції як об'єкти першого рівня, об'єкти як списки, каррінг (currying), анонімні функції, замикання (closures) — що додає мові додаткову гнучкість.

JavaScript має C-подібний синтаксис, але в порівнянні з мовою Сі має такі корінні відмінності:

- об'єкти, з можливістю інтроспекції і динамічної зміни типу через механізм прототипів
- функції як об'єкти першого класу
- обробка винятків
- автоматичне приведення типів
- автоматичне прибирання сміття

- анонімні функції

Семантика мови схожа з семантикою мови Self.

Специфікація ECMAScript є стандартизованою специфікацією мови сценаріїв, розробленої Бренданом Айхом з Netscape ; спочатку він був названий Mocha, пізніше LiveScript, і нарешті JavaScript. У грудні 1995 року Sun Microsystems і Netscape оголосили JavaScript у прес-релізі. Перше видання ECMA-262 було прийнято Генеральною Асамблеєю Ecma у червні 1997 року.[13] З того часу було видано кілька видань мовного стандарту. Назва "ECMAScript" була компромісом між організаціями, що займаються стандартизацією мови, особливо Netscape і Microsoft, чиї суперечки домінували на ранніх сесіях стандартів. Кожен коментував, що "ECMAScript завжди був небажаною торговою назвою, що звучить як шкірна хвороба ".

Хоча як JavaScript, так і JScript прагнуть бути сумісними з ECMAScript, вони також надають додаткові функції, не описані в специфікаціях ECMA.

При розробці великих і нетривіальних веб-застосунків з використанням JavaScript, критично важливим є доступ до інструментів зневадження. Оскільки браузери від різних виробників дещо відрізняються у поведінці (в тому числі і в Об'єктній Моделі Документа, треба мати в руках зневаджувачі для кожного браузера, якщо веб-застосунок орієнтовано на нього.

Firefox, Google Chrome, Opera та Safari мають вбудовані зневаджувачі під себе.

Internet Explorer має три зневаджувачі для себе: Microsoft Visual Studio є найпотужнішим з цих трьох, слідом йде Microsoft Script Editor (компонента Microsoft Office[2]), також існує безкоштовний Microsoft Script Debugger з базовими функціями. Веб-застосунки для Firefox допоможе вдосконалити додаток Firebug (зручно вбудований безпосередньо в браузер), або давніший зневаджувач Venkman, котрий також працює з браузером Mozilla. Drosera — це зневаджувач з WebKit engine[3], що супроводжує Apple Safari.

Також існують кілька інструментів, як вільних, наприклад JSLint[4], інструмент перевірки якості коду, що сканує JavaScript програму, шукаючи проблеми коду, так і комерційних продуктів типу інструменту з назвою JavaScript Debugger.

Оскільки ECMAScript є інтерпретатором, без суворої типизації, і може виконуватися в різних середовищах, кожне зі своїми власними особливостями сумісності, програміст має бути дуже уважним, і повинен перевіряти, що його код виконується як очікується в широкому переліку можливих конфігурацій. Дуже часто трапляються випадки, коли скрипт, що чудово працює в одному середовищі, видає некоректні результати в іншому.

Кожен блок сценарію інтерпретатор розбирає окремо. На веб-сторінках, коли треба комбінувати блоки JavaScript та HTML, синтаксичні помилки знайти простіше, якщо тримати функції сценарію в окремому блоці коду, або (ще краще) використовувати багато малих пов'язаних `.js` файлів. В такий спосіб синтаксична помилка не спричинятиме «падіння» цілої сторінки, і можна надати допомогу, елегантно вийшовши зі сторінки.

Для серверних проектів node.js можна використовувати інтегроване середовище розробки WebStorm.

3.8 Вимоги до системи

Для адекватної роботи сервісу рекомендовано використовувати браузери Google Chrome 6.0+, Opera 9.6+, Internet Explorer 9.0+, Safari 3.1+ або Mozilla FireFox 4.0+.[9] Для запуску програми на вашому персональному комп’ютері має бути встановлена версія Node 6.0+, Angular CLI та будь-який з браузерів, наведених вище.

3.9 Висновки до розділу

В цьому розділі були розглянуті засоби, які були використані для розробки сервісу. Так як розроблена система є web-сервісом і призначена для роботи з нею

через браузер, було обрано єдиний варіант мови розмітки HTML, каскадні стилі CSS та мова програмування Javascript для клієнтської сторони і мова програмування Node.js для серверної. У випадку з Node.js був вибір між іншими мовами, але Node.js ідеально підходить для невеликих додатків та використовує в собі мову Javascript, тобто розробка на клієнтській і серверній стороні велась з одною структурою і стилем написання коду. Але для клієнтської сторони ще було використано фреймворк Angular, що надає додаткові можливості, полегшує роботу, ховаючи багато чого під капотом і дозволяє використовувати мову програмування Typescript замість Javascript і препроцессор Sass замість Css. Всі ці технології покращують систематизацію коду, оптимізують додаток і є прогресивними в наш час. Через всі вищенаведені фактори було обрано саме ці технології.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

В цьому розділі ми розглянемо опис розробленого програмного продукту та особливості програмної реалізації модулів розробленої системи:

1. Побудова дна
2. Побудова зображення хвилі
3. Генерація безпосередньо гідроакустичного сигналу

4.1 Початкові дані системи

Вхідними даними для генерації гідроакустичного сигналу є такі параметри, як:

1. Координати та глибина гідрофона (ГАС);
2. Характеристики об’єкту шуму (рухомий морський об’єкт)
   - 2.1 Стартове та кінцеве положення;
   - 2.2 Глибина;
   - 2.3 Швидкість;
   - 2.4 Амплітуда коливань;
   - 2.5 Частота хвил;
3. Геометричні розміри хвилі;
4. Час моделювання;
5. Частота коливання хвиль кожного об’єкта;
6. Амплітуда коливань хвилі;

Рисунок 4.1 — Інтерфейс внесення вхідні даних в систему

Усі початкові дані вводяться в систему (Рисунок 4.1) та використовуються у майбутніх розрахунках для постройки графіків, які визначають відстань від морського об’єкта до гідрофона в кожен момент часу та відображають хвильову картинку звукових сигналів від морського об’єкта до гідрофона, враховуючи при цьому геометрію морського дна в кожен момент часу та для генерації кінцевого гідроакустичного сигналу, що є головною задачею роботи програмного продукту.

4.2 Загальний опис розробленої системи

Розглянемо задачу з генерації гідроакустичного сигналу променевим методом покроково.

Для початку визначимо початкові вхідні дані, необхідні нам для подальших розрахунків. Щоб виконати цю задачу потрібно продумати і створити зручний та простий користувацький інтерфейс. За допомогою нього користувач програми зможе задати необхідні початкові вхідні параметри для моделювання.

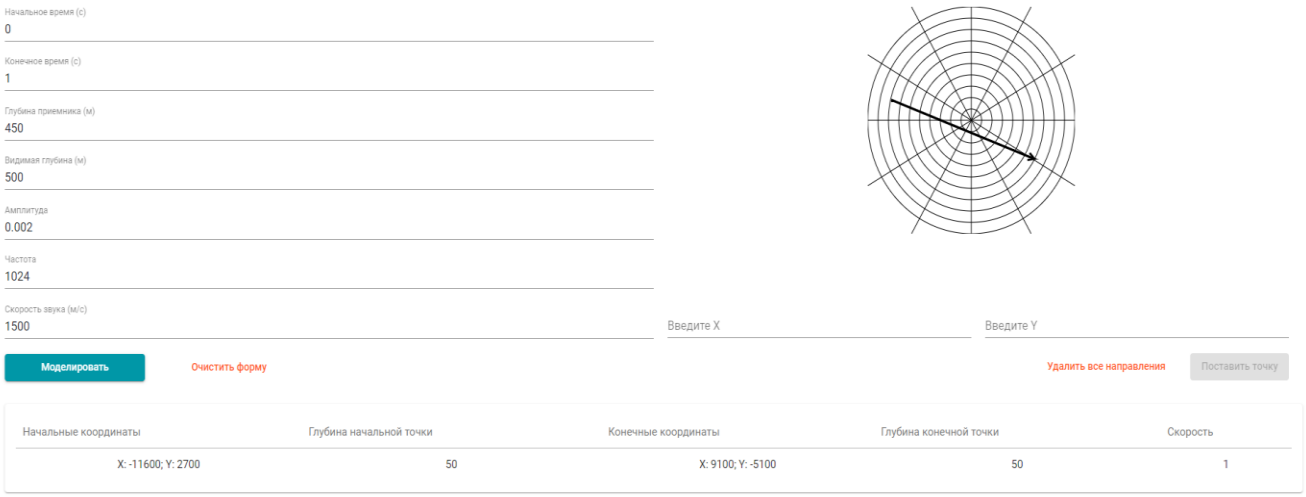

Рисунок 4.2 — Користувацький інтерфейс головного екрану програмного продукту

Інтерфейс має включати в себе акваторію (Рисунок 4.2), що утворена комбінацією концентричних кіл та осей координат, щоб полегшити користувачу розуміння масштабування акваторії, на якій будуть будуватись початкові та кінцеві координати траєкторій руху об’єктів; елементи керування для вводу необхідних вхідних даних для розрахунків, таких як: глибина занурення гідрофона, час моделювання та координати кінцевих точок дна взятого для моделювання. Інтерфейс також включає в себе таблицю, в якій міститься список створених морських об’єктів з їх параметрами, та кнопка для відкриття діалогового вікна з графіками, які можна переглянути після розрахунків.

Наступним кроком було визначення масштабу сітки нашої акваторії, було прийнято рішення розрахувати масштаб так, що центральне коло має радіус 1500м. (Рисунок 4.3)

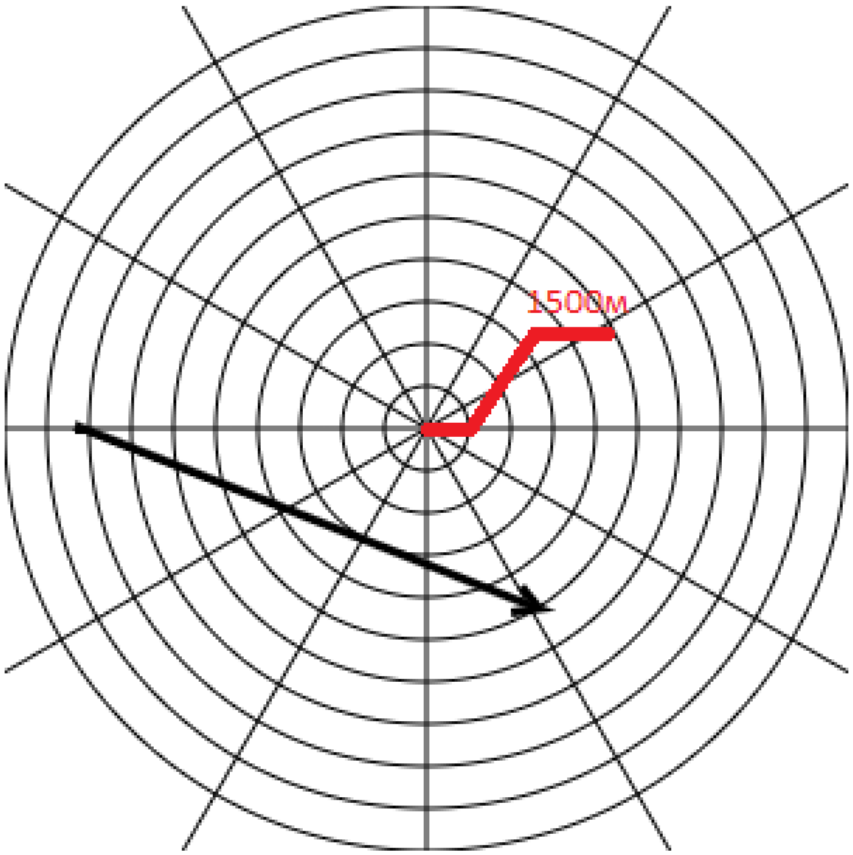

Рисунок 4.3 — Масштаб сітки акваторії

Для зручності нанесення об’єктів на акваторію при необхідності розрахунків з точними даними необхідно було реалізувати можливість введення даних через поля вводу. (Рисунок 4.4)

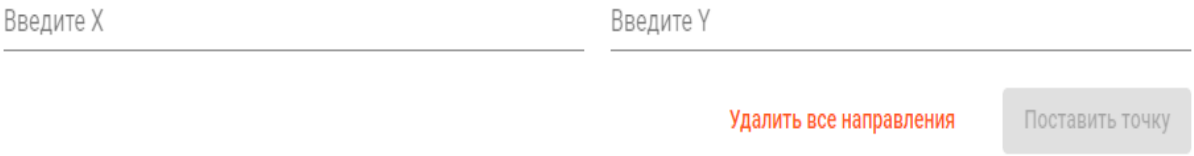

Рисунок 4.4 — Інтерфейс головного екрану програмної системи

Також при постановці кожної точки спливає діалогове вікно, в якому необхідно вказати глибину точки. (Рисунок 4.5)

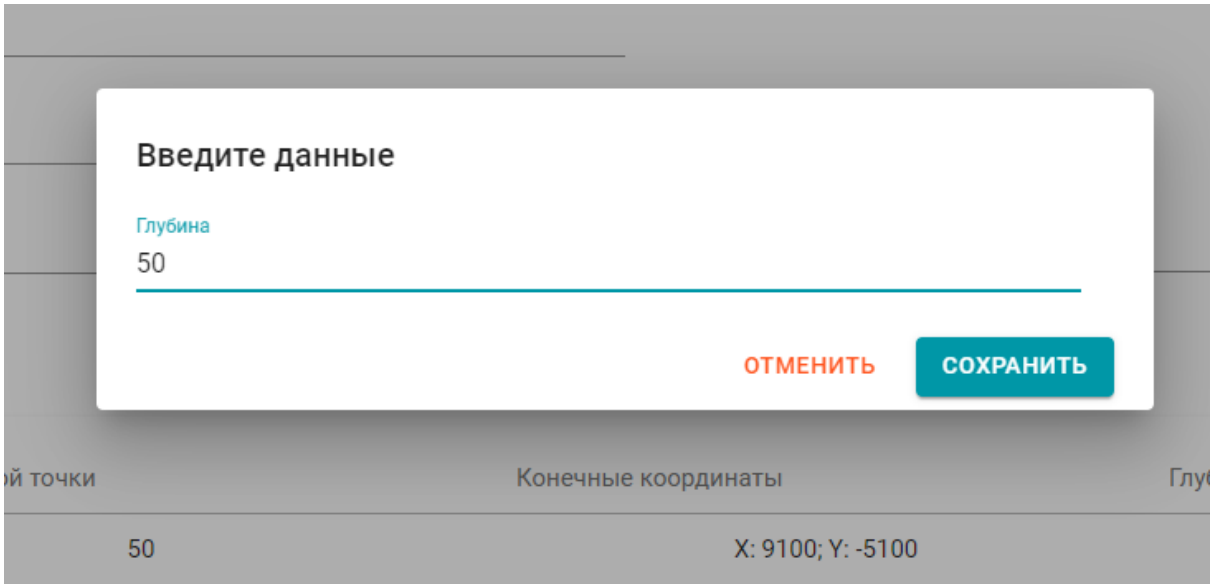

Рисунок 4.5 — Діалогове вікно при підтвердженні початкової точки

Якщо ми задаємо кінцеву точку направлення морського об’єкту, то крім глибини необхідно ввести ще й його швидкість. (Рисунок 4.6)

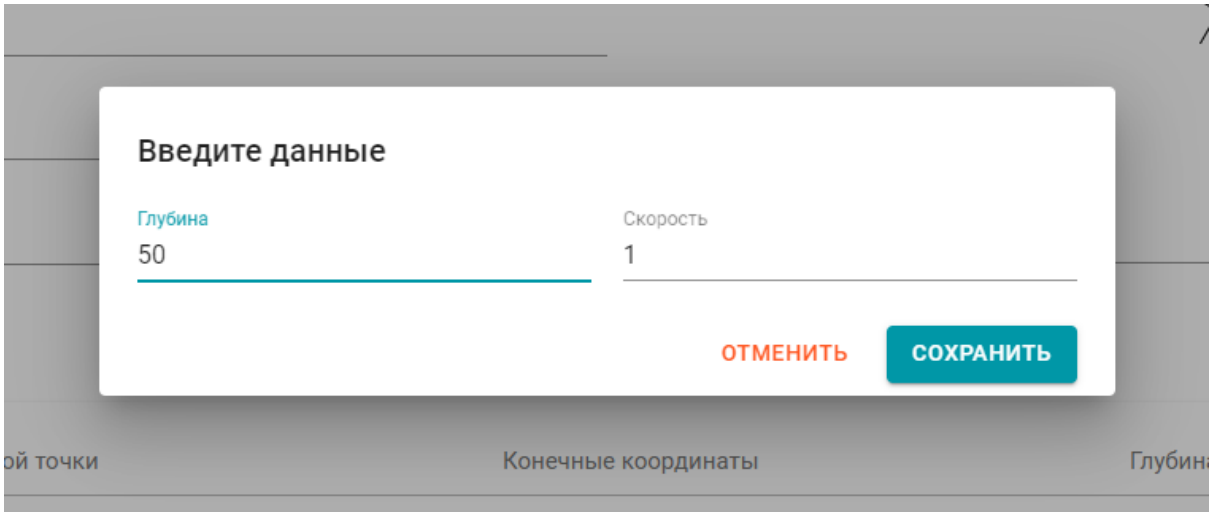

Рисунок 4.6 — Діалогове вікно при підтвердженні кінцевої точки

Далі потрібно розробити алгоритм для розрахунку всіх параметрів об’єктів та водного середовища, такі як: глибина дна, пересування об’єкта за його траєкторією, отримання фаз, довжини хвиль, розрахунок найбільш можливого занурення морських об’єктів відносно глибини водного середовища, методи роботи з векторами (множення, нормалізація, додавання, отримання орт-вектора), методи розв’язування систем лінійних рівнянь, щоб мати можливість знайти проекції точок

пересування об’єкту на дно середовища, алгоритм відбивання променів хвиль від поверхні та дна та інші.

Далі, маючи всі необхідні початкові дані, ми можемо отримати кінцеву точку шляху об’єкта та його розташування в будь-який момент часу. Це дасть нам можливість, з допомогою алгоритмів розрахунку даних, реалізованих завчасно, згенерувати гідроакустичний сигнал та записати в файл розширення dat.

Алгоритм генерації гідроаукустичного сигналу буде мати такий вигляд:

1. Задання початкових параметрів для моделювання
2. Для кожного моменту часу(крок часу між переглядами стану водного об’єкта) порахувати фазу тиску, швидкості та амплітуду на приймачі.
3. Скласти та визначити фазу тиску, швидкості та амплітуду по променю векторно.
4. Порахувати суперпозицію швидкостей та тисків на приймачі.

Після отримання показників тиску та швидкості хвилі, їх значення на кожному кроці часу потрібно масштабувати до значень цілих чисел у діапазоні -16000-16000 для більш крупної вибірки. Значення, які ми отримаємо після всіх розрахунків необхідно перевести в бінарний формат та записати у файл розширення .dat у послідовності p, vz, vx, vy. Створений файл і є згенеровним гідроакустичним сигналом.

4.3 Відбивання хвилі від поверхонь

Перед генерацією сигналу потрібно отримати характеристики деякої кількості хвиль, що доходять до гідрофону, рухаючись безпосередньо прямо від джерела шуму до гідрофона, або відбиваючись від дна чи поверхні водного середовища. Алгоритм для знаходження траєкторії хвиль базується на фізичних законах про відбивання світла. Так як за цими законами кут відбиття світла рівний куту падіння, це дозволяє нам застосовувати перпендикулярне проектування, щоб знайти падаючі промені. (Рисунок 4.7)

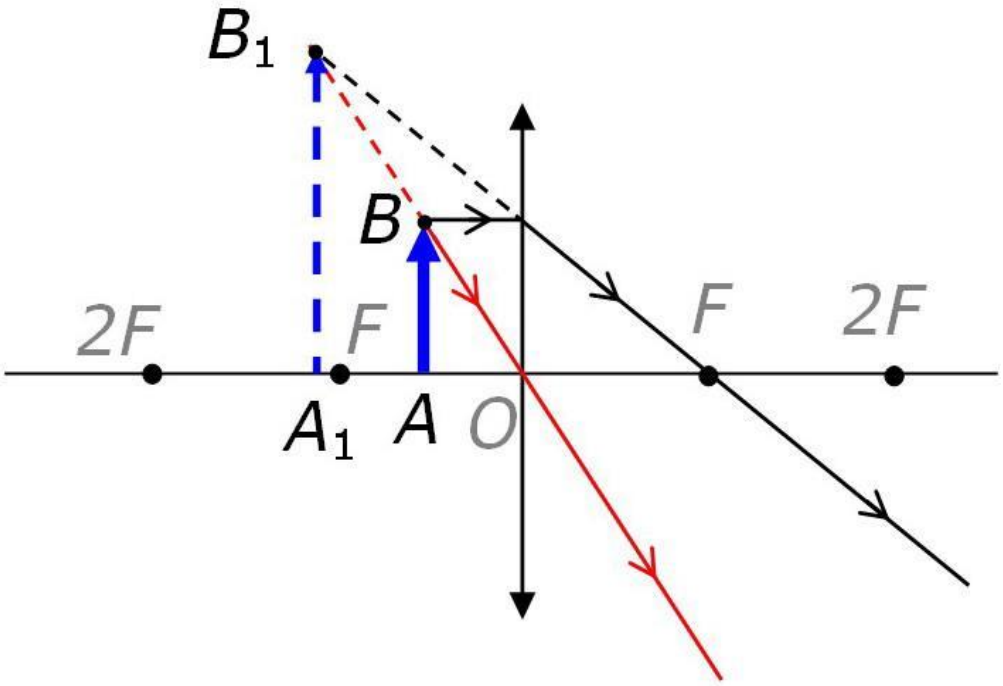

Рисунок 4.7 — Побудова уявних зображень

Цей алгоритм оснований на понятті уявних зображень. Оптичне зображення, створене променями, які не перетинаються між собою, а тільки їх уявні продовження перетинаються. Прикладом такого зображення є відображення предметів через дзеркало.

Головна суть цього алгоритму полягає у тому, щоб будувати уявні точки ще й за межами акваторії. Побудувавши такі точки в нас буде можливість точно визначити куди влучить промінь хвилі, що буде опиниться в цій точці. Алгоритм побудови точок:

1. Спроектуємо точку гідрофона перпендикулярно до якоїсь з поверхонь.
2. Випустимо хвилю з точки нашого об’єкта в уявну сформовану точку
3. Відбившись від поверхні або дна, промінь попаде рівно в гідрофон та ми отримаємо можливу траєкторії руху хвилі

Алгоритм побудови уявного зображення, описаний вище, проілюстровано схематично на рисунку 4.8.

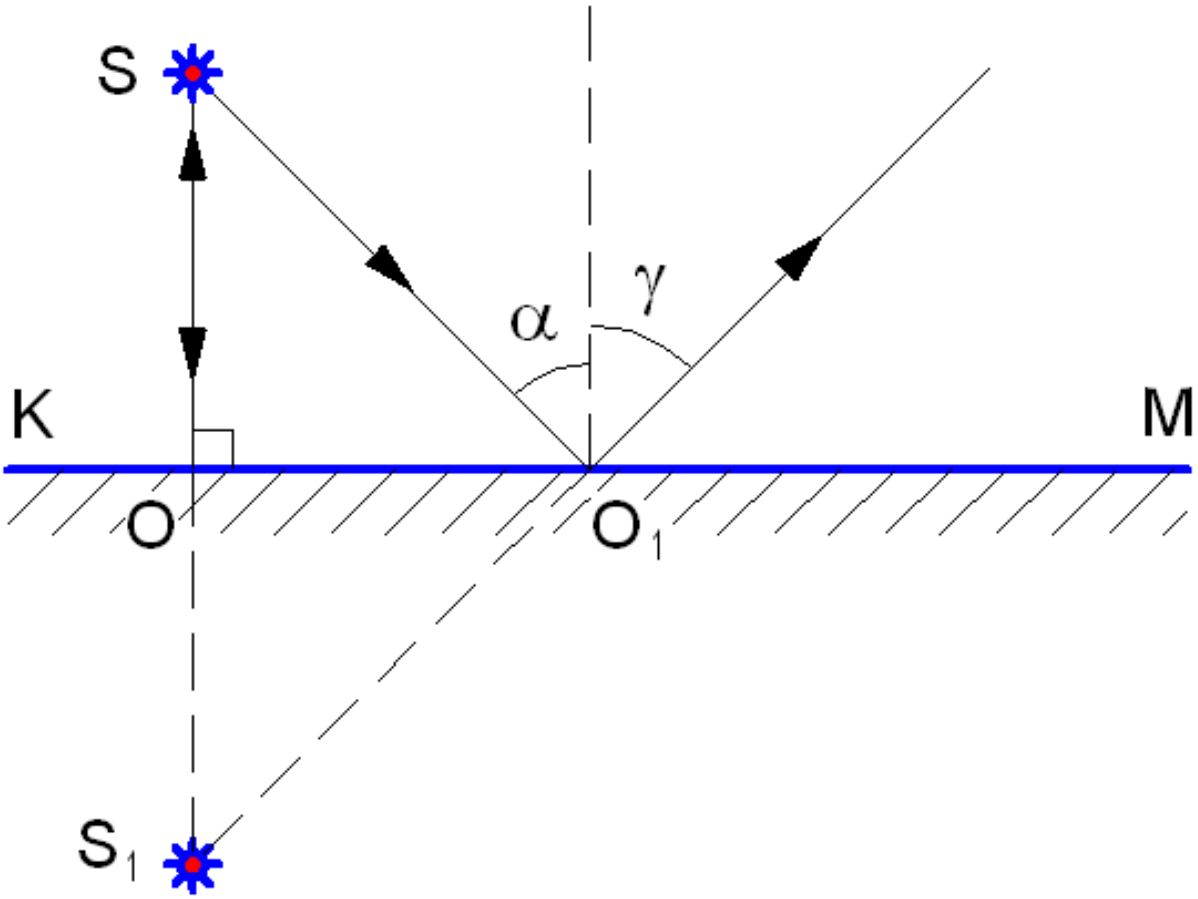

Рисунок 4.8 — Побудови траєкторії руху променя

Також існує й інший спосіб побудови траєкторії руху хвиль. Він полягає в переборі кутів під якими випущений промінь влучить у гідрофон. Цей метод є легшим для реалізації, тому що все що треба, це покроково змінювати кут виходу хвилі з об’єкта та перевіряти, чи влучила ця хвиля у гідрофон. Але цей метод гірше ніж оптичний метод по кількості ітерацій виконуваних операцій, відповідно і у швидкості обробки даних та розрахунків. Через це і було прийняте рішення обрати метод, що заснований на оптичних законах. (Р)

Також, за допомогою оптичного метода ми будемо мати можливість заздалегідь встановити максимальну кількість ітерацій відбиття від поверхні, так як хвилі затухають з часом і не будуть відбиватись нескінченність разів, що є ще однією великою перевагою цього метода. Розрахувати цю кількість просто, кількість разів хвиля, що хвиля повинна відбитися від поверхні, стільки ж перпендикулярних проекцій уявної точки і необхідно зробити.

4.4 Генерація сигналу

І останньою проблемою у розв'язанні нашої задачі є пошук швидкості руху та тиску променя в заданій точці (на гідрофоні). Для цього потрібно знайти головні показники променя у цій точці, такі як: амплітуду і фазу тиску, кутову швидкість, хвильовий вектор, його довжину та фазу хвилі. Ще для обчислення нам знадобляться такі акваторні параметри як: швидкість звуку та густина середовища.

Нижче наведено формулу, яка дозволяє розрахувати швидкість променя в заданій точці:

$$v(r,t) = \omega\delta_m \cos\left(kr - \omega t - \varphi_{\delta,0} + \frac{\pi}{2}\right) = v_m \cos(kr - \omega t + \varphi_{v,0}) \quad (4.1)$$

Основні величини формули 4.3.1:

1. ω — кутова швидкість;
2. δ_m — амплітуда променя;
3. k — довжина променевого вектора;
4. r — відстань, що пройшов промінь до гідрофона;
5. $\varphi_{\delta,0}$ — початкова фаза променя;
6. $\varphi_{v,0}$ — зміщення фази в кінцевій точці руху;

Спочатку потрібно визначити кутову швидкість, що знадобиться нам для розрахунку швидкості променя і тиску.

$$\omega = 2\pi\vartheta \quad (4.2)$$

ϑ — частота променя, що задається окремо для кожного променя перед початком моделювання.

Іншою важливою величиною для розрахунків є хвильовий вектор. Він направлений перпендикулярно до фазового фронту променя і розраховується за

формулою нижче:

$$k = \frac{2\pi}{\lambda} \tag{4.3}$$

Після розрахування цих величин, в нас буде можливість розрахувати довжину вектора швидкості у кінцевій точці руху вектора. Тільки для отримання гідроакустичного сигналу нам ще потрібно розбити отримане значення на величини vx, vy, vz векторним шляхом. Нам відомо, що вектор змінюється на кожному кроці на деяку фазу, тож щоб знайти його направленість необхідно порахувати фазу в той момент, коли він буде в кінцевій точці свого руху. Скористаємось наступною формулою:

$$\varphi = \ t * 2\pi\vartheta \tag{4.4}$$

Розрахувавши фазу, відхилимо вектор відносно початкової фази променя на отриману фазу по формулі 4.4. та розкладамо отриманий вектор на vx, vy, vz векторним шляхом. Запишемо ці значення.

Тепер розглянемо формулу знаходження тиску променя в заданій точці:

$$p(r,t) = pc^2 k\delta_m \cos\left(kr - \ \omega t - \ \varphi_{\delta,0} + \frac{\pi}{2}\right) = \ p_m \cos(kr - \ \omega t + \ \varphi_{p,0}) \tag{4.5}$$

Розглянемо основні величини формули 4.5

1. ω — кутова швидкість;
2. δ_m — амплітуда променя;
3. k — довжина променевого вектора;
4. r — відстань, яку пройшов промінь до гідрофона;
5. $\varphi_{\delta,0}$ — початкова фаза променя;
6. $\varphi_{v,0}$ — зміщення фази в кінцевій точці руху;

7. p — густина водного середовища;
8. c — швидкість розповсюдження шуму в водному середовищі;

З формул 4.1 та 4.5випливає, що як тиск так і швидкість залежать від двох основних параметрів: часу руху променя (t) та довжини шляху променя (r).

Також потрібно зауважити, що швидкість розповсюдження шуму у воді набагато більша ніж в повітрі та складає десь 1500 м\с, а в повітрі всього 300 м\с.

Після того як ми знайшли p, vx, vy та vz, построїмо для них окремі графіки, що будуть являти собою залежність цих параметрів від кожного моменту часу та зможемо візуально подивитись на модуляцію гідроакустичного сигналу. (Рисунки 4.9)

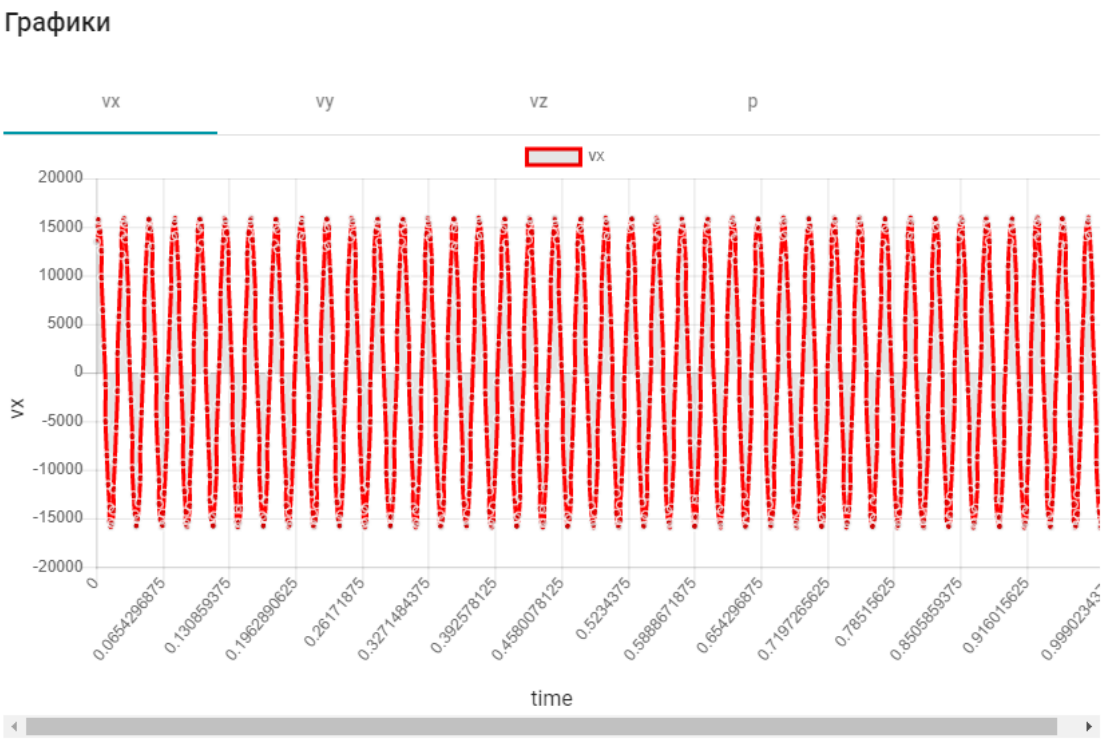

Рисунок 4.9 — Візуальне представлення відношення vx до часу

Після отримання необхідних нам показників (тиску та швидкості променя) необхідно масштабувати їй значення для кожного моменту часу до значень цілих чисел в діапазоні -16000-16000 для більш крупної вибірки. Фінально отримані значення потрібно перевести у бінарний формат та записати в файл розширення .dat у послідовності p, vz, vx, vy. Даний файл і буде шуканим гідроакустичним сигналом.

4.5 Висновки до розділу

В даному розділі було описано вигляд програмного продукту, його вхідні дані, а також головні частини логіки роботи системи. Розділи 4.3та 4.4 показують основні елементи системи;;

1. Побудова алгоритму для відбивання хвиль від певерхні та дна водного середовища і програмна його реалізація;
2. Безпосередньо генерація гідроакустичного сигналу і візуалізація характеристик сигналу p, vx, vy і vz в графічному вигляді.

В кожному з підрозділів була ретельно розглянута розробка алгоритмів для вирішення поставлених нам задач та наведені приклади роботи вже реалізованих алгоритмів, також наведені формули для розрахунків та їх пояснення.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Інтерфейс програми виконано за допомогою HTML розмітки, CSS стилів та мови програмування javascript з використанням фреймворка Angular, що дозволяє використовувати препроцессор SASS замість CSS та Typescript замість Javascript, які дають ширші можливості. Він включає в себе багато елементів керування для введення даних, діалогових вікон, вкладок і графіків для візуалізації інформації про рухомі морські об’єкти та водне середовище. (Рисунок 5.1)

Рисунок 5.1. — Головний екран програми з введеними початковими даними та відміченим рухомим морським об’єктом на акваторії

Розпочну опис інтерфейсу з акваторії, яка є сіткою з концентричних кіл та системою координат. Даним компонентом користувацького інтерфейсу можна керувати за допомогою лівої клавіші миші. Натискання на клавіші миші призведе до встановлення початкової позиції рухомого морського об’єкта та відкриття модального вікна, в якому необхідно буде ввести глибину (z-координата об’єкта) початкової точки водного об’єкта (Рисунок 5.2).

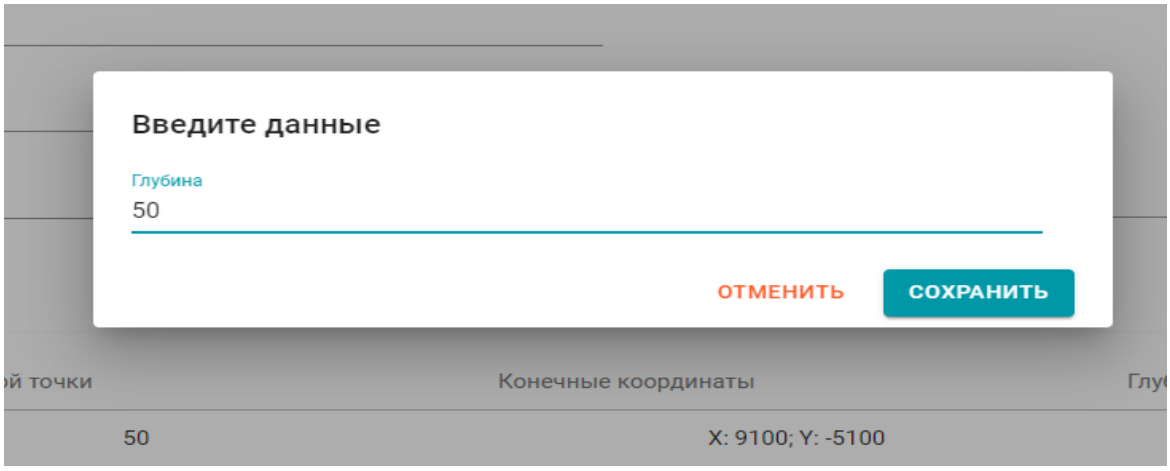

Рисунок 5.2. — Діалогове вікно після задання початкової точки направлення об’єкта

При наступному натисканні на праву клавішу буде встановлена кінцева точка направлення руху об’єкта, а також знову спричинить виклик діалогового вікна, в якому користувачу необхідно буде ввести не тільки глибину занурення об’єкта (z-координату), а ще його швидкість. (Рисунок 5.3).

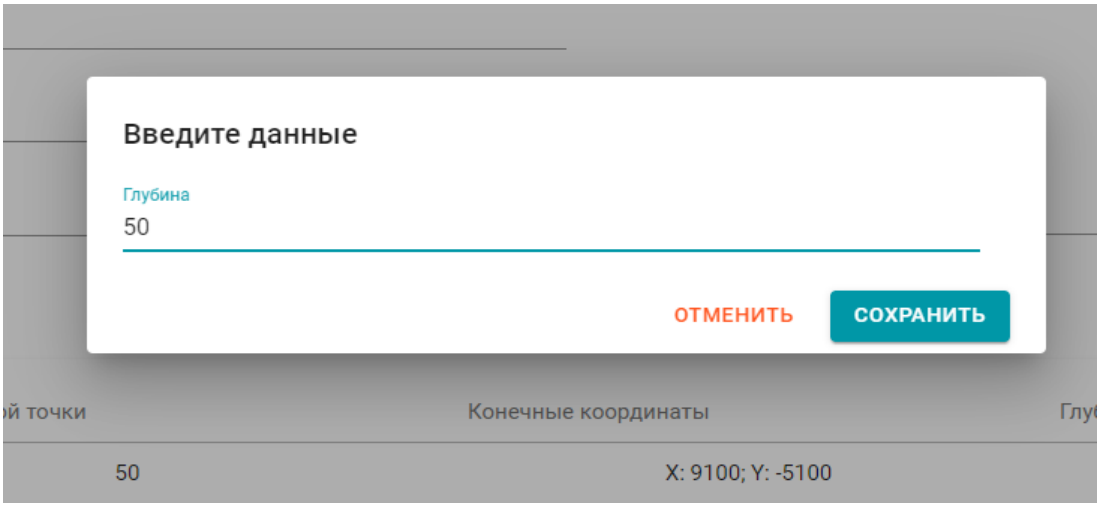

Рисунок 5.3 — Діалогове вікно після задання кінцевої точки направлення об’єкта

У лівій частині користувацького інтерфейсу знаходиться таблиця з списком усіх морських об’єктів, які створив користувач, їх початковою позицією, точкою направленості, глибиною та швидкістю (Рисунок 5.3).

Також є можливість задати направлення через поля вводу для X та Y

координати. Це необхідний і зручний функціонал коли необхідно ввести точні показники. (Рисунок 5.4)

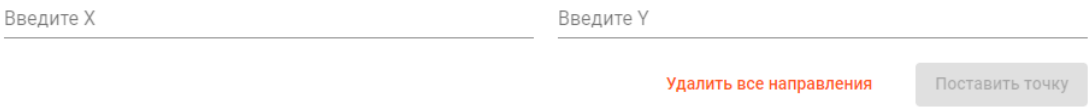

Рисунок 5.4 — Інтерфейс для введення координат рухомих об’єктів з клавіатури

Крім того є можливість видали всі відмічені вже направлення по натисканню однієї кнопки, що дозволяє розпочати генерацію наступного сигналу з іншими даними без перезавантаження сторінки.

Введені дані про направлення морських об’єктів зберігаються та відображаються в таблиці, наведеній нижче. (Рисунок 5.5)

Начальные координаты	Глубина начальной точки	Конечные координаты	Глубина конечной точки	Скорость
X: -9200; Y: -200	50	X: 3900; Y: -5400	50	1
X: 6300; Y: 1600	50	X: -11700; Y: -6900	54	1
X: -4700; Y: -10300	50	X: -7400; Y: 2000	52	8
X: -2200; Y: 8800	60	X: 8300; Y: 5700	360	100
X: -6900; Y: 4900	50	X: 2800; Y: -12200	50	1

Рисунок 5.5 — Список водних об’єктів доданих користувачем та їх параметрів

У лівій частині користувацького інтерфейсу з формою для вводу початкових даних. (Рисунок 5.6)

Вхідні дані:

- Початковий час
- Кінцевий час
- Глибина гідрофона
- Глибина дна
- Амплітуда коливань променя

- Частота коливань
- Швидкість звуку

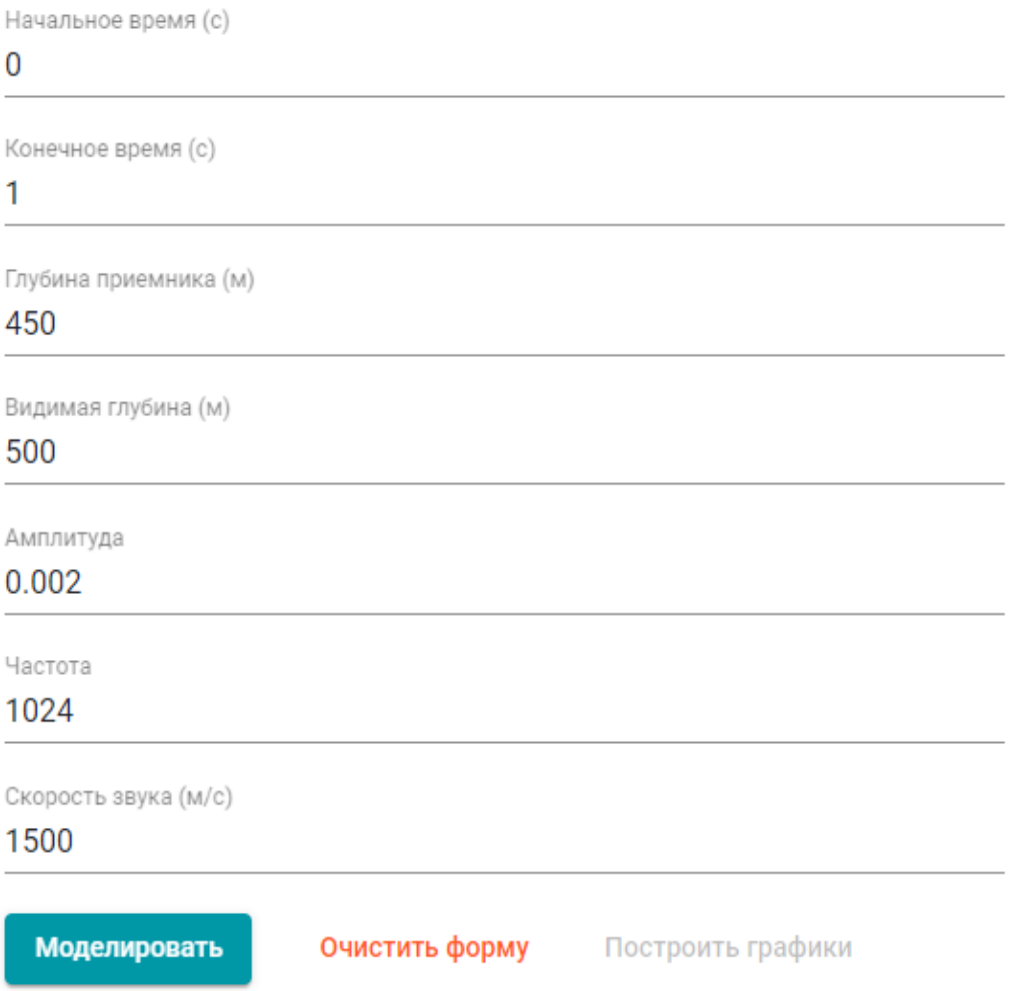

Рисунок 5.6 — Форма для вводу вхідних даних

Ще присутня кнопка для очистки кнопка для очистки всіх полів. Це дозволяє швидко і зручно почати заповнення нових даних для генерації наступного гідроакустичного сигналу.

На поля форми накладена валідація тому при введенні невірних даних кнопка відповідальна за моделювання буде недоступною, а поле, в якому введено невірне значення або не введено нічого буде підкреслюватись червоним кольором. (Рисунок 5.7)

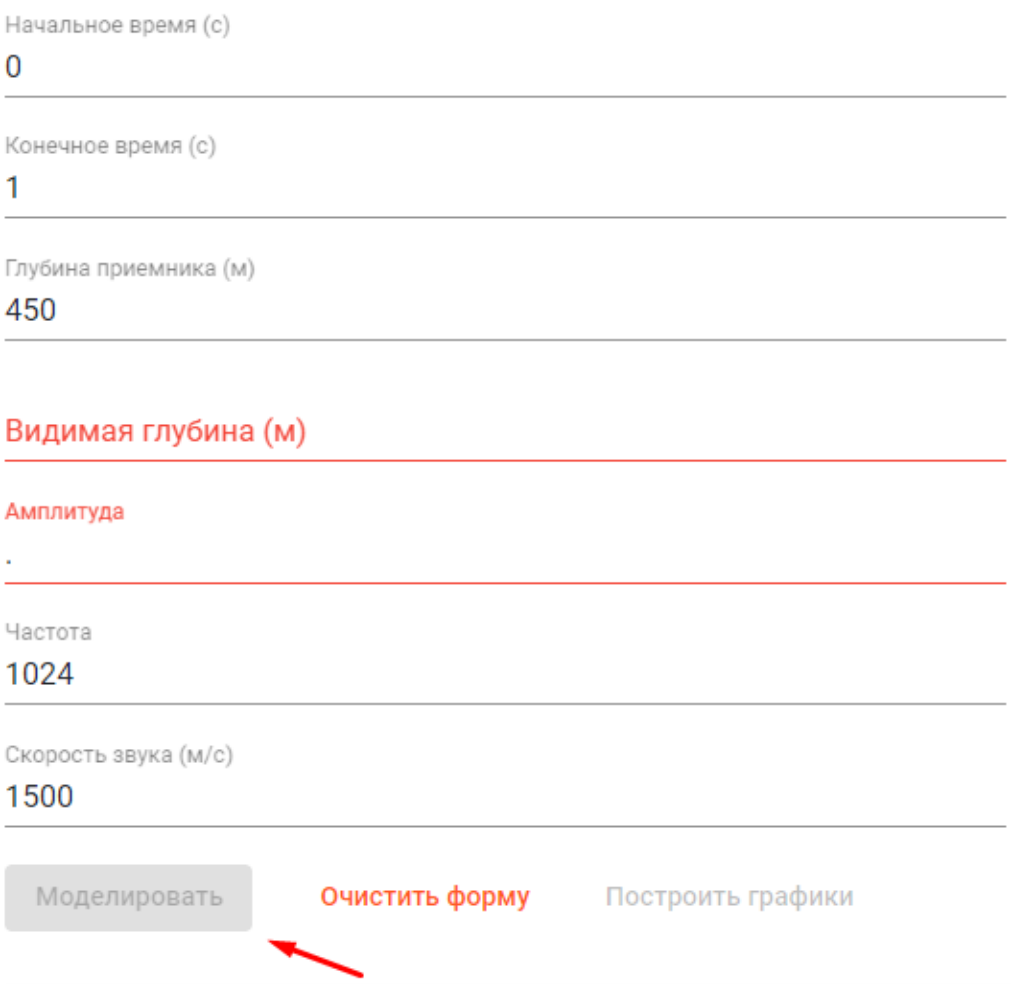

Рисунок 5.7 — Невірно заповнена форма для вводу початкових даних

Після правильного введення всіх необхідних даних кнопка для моделювання буде активною і після її натискання почнеться розрахунок на стороні сервера. Впродовж цього часу з самого верху веб-сторінки почне рухатись progress-bar (полоса загрузки), що дозволяє користувачу зрозуміти, що програма в процесі обробки, а не просто проігнорувала його дії. (Рисунок 5.8) Це корисно когда розмова йде про сигнали великих розмірів і громіздкі розрахунки які можуть займати час.

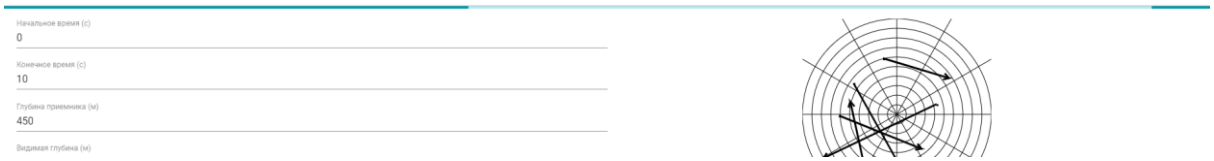

Рисунок 5.8 — Полоса загрузки (progress-bar), що з’являється під час розрахунків

Якщо розрахунки було завершено успішно нам випадає діалогове вікно про успішно згенерований сигнал і пропозицію його завантажити. (Рисунок 5.9)

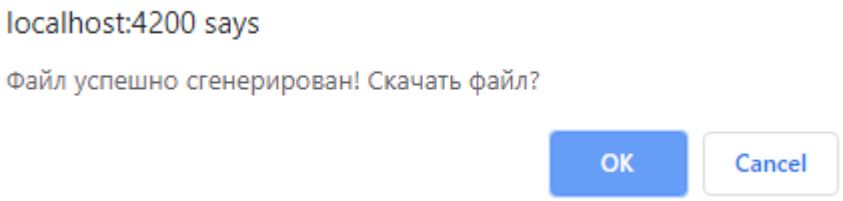

Рисунок 5.9 — Діалогове вікно про успішне закінчення генерації файлу

Після натискання на кнопку відміни, дані про сигнал збережуться і користувач зможе подивитись графіки на основі цих даних. При натисканні на кнопку підтвердження дані також збережуться для візуалізації в графіки, але крім того відкриється стандартне браузерне діалогове вікно скачування і після вибору назви і шляху для збереження нам завантажиться файл dat розширення, який і являє собою наш гідроакустичний сигнал. (Рисунок 5.10).

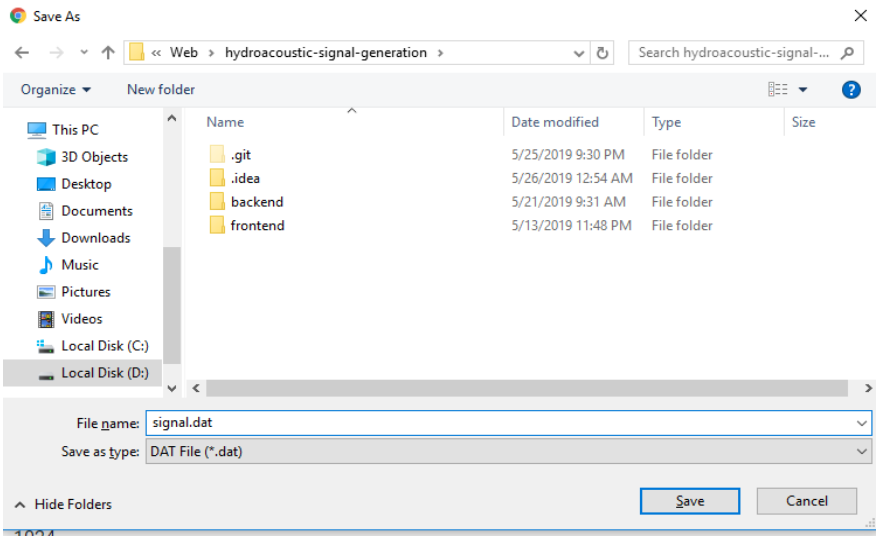

Рисунок 5.10 — Діалогове вікно збереження файлу

Збережений файл можемо прослухати в програмі, що називається Audacity. (Рисунок 5.11)

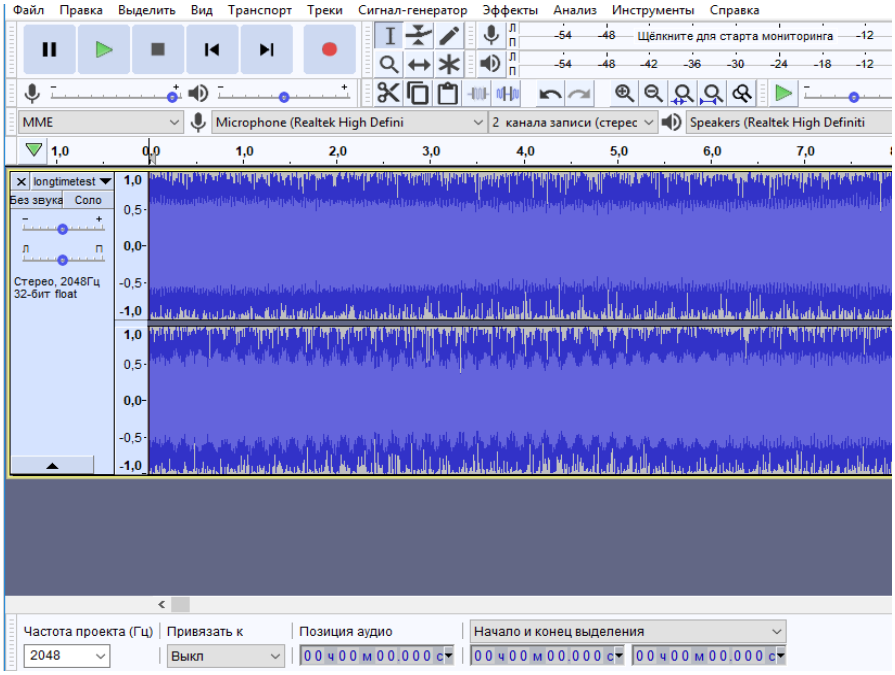

Рисунок 5.11 — Программа Audacity з імпортованим сигналом для його прослуховування

І остання частина програми — можливість переглянути графіки відношення тиску та швидкостей до часу. Для цього після того як сигнал згенерується необхідно натиснути на кнопку “Показать графики”, в результаті чого відкриється діалогове вікно з чотирма вкладками, на кожну вкладку свій графік.

Побудуємо графік відношення векторної швидкості по X до часу (Рисунок 5.12).

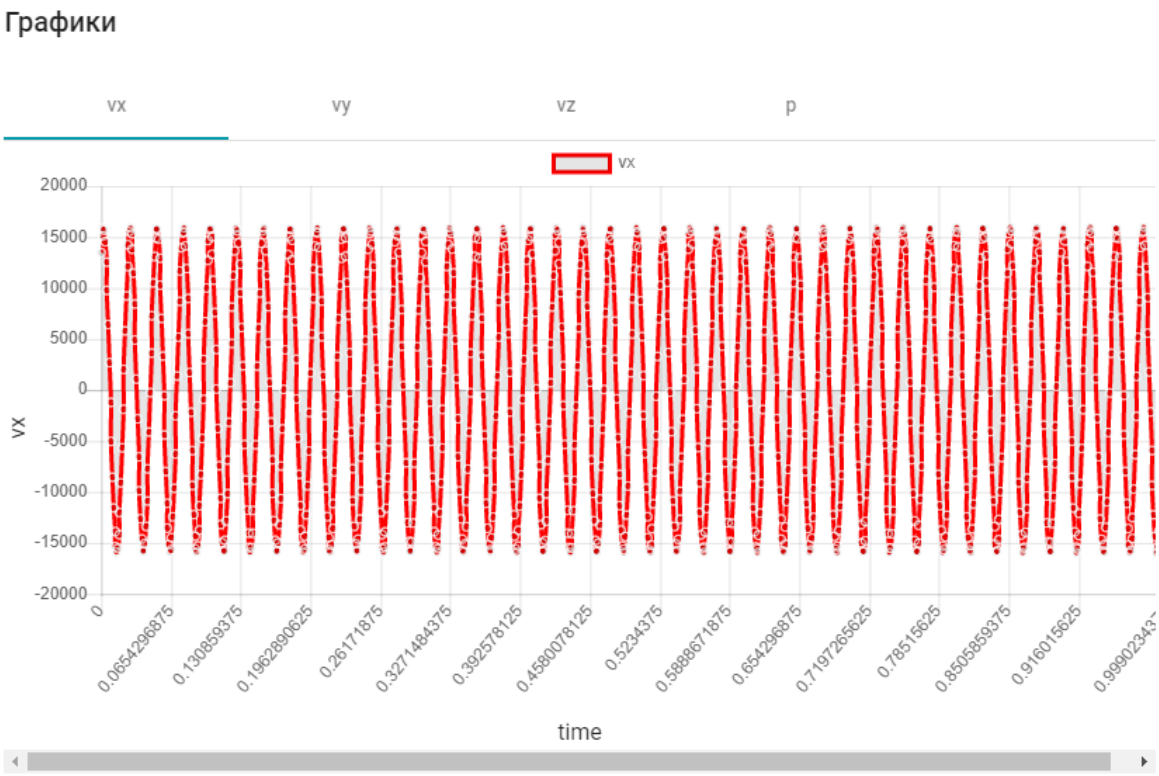

Рисунок 5.12 — Приклад графіка з відношенням векторної швидкості по X до часу

Переключимось на вкладку VY та подивимся на візуалізацію відношення векторної швидкості по Y до часу. (Рисунок 5.13)

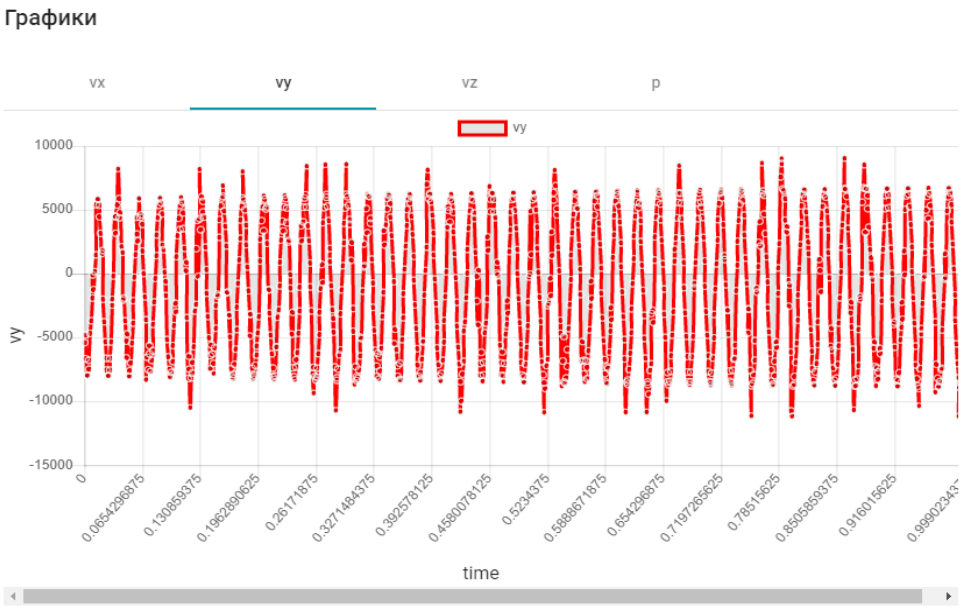

Рисунок 5.13 — Приклад графіка з відношенням векторної швидкості по Y до часу

Розглянемо також графік з відношенням векторної швидкості по Z до часу. (Рисунок 5.14)

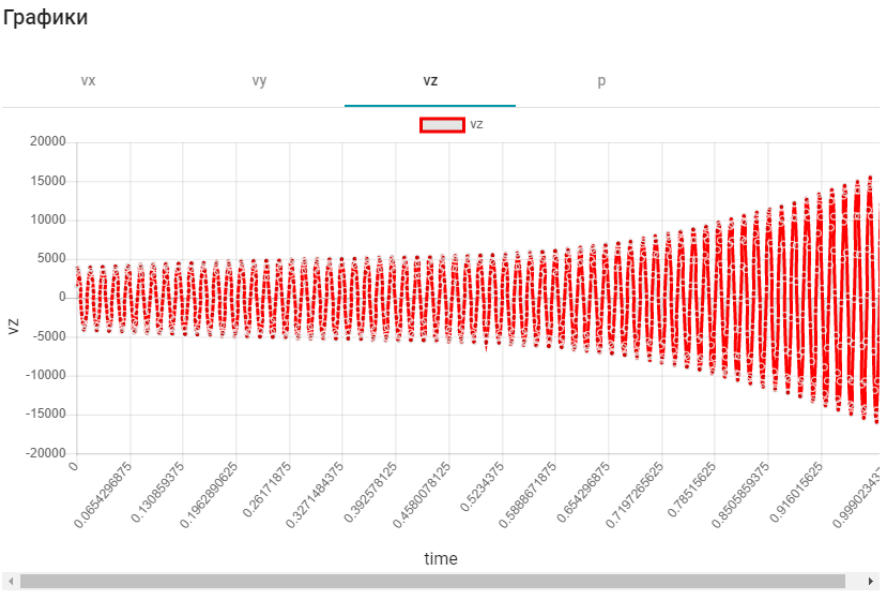

Рисунок 5.14 — Приклад графіка з відношенням векторної швидкості по Z до часу

І візуалізацію останнього графіка з відношенням тиску до часу можна подивитись на рисунку 5.15.

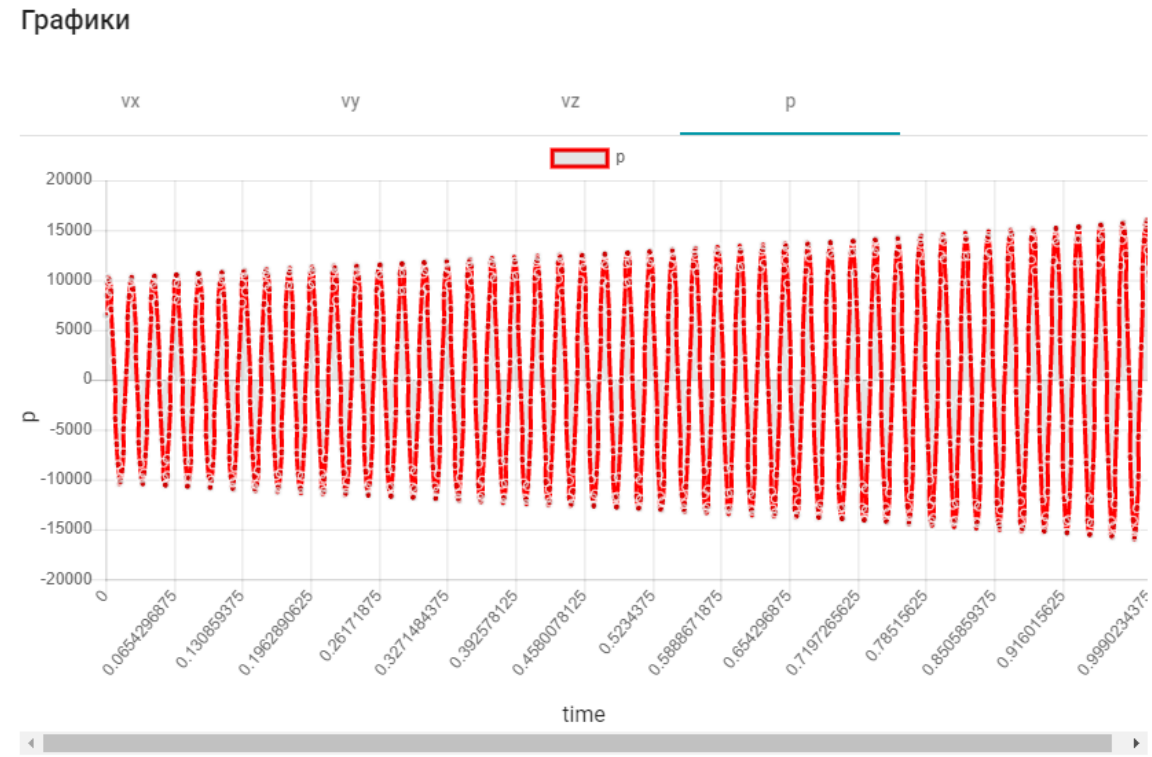

Рисунок 5.13 — Приклад графіка з відношенням тиску до часу

5.1 Висновки до розділу

В цьому розділі було наведено інформацію про алгоритм користування сервісом та всі його можливості, що в свою чергу дозволить користувачу легше і швидше зрозуміти принципи роботи з програмною системою. В допомогу до розуміння користувача були подані скріншоти з покроковою інструкцією користування програмою і всіма її модулями та компонентами від введення початкових даних до прослуховування сигналу.

ВИСНОВКИ

По ходу виконання дипломної роботи було розроблено веб-сервіс для генерації гідроакустичних сигналів водних рухомих об’єктів променевим методом.

Вхідні дані програма отримує через компоненти керування головним інтерфейсом, за допомогою яких користувач може вводити необхідні дані для моделювання.

Під час виконання дипломної роботи було виконано наступні завдання:

1. Був розроблений алгоритм для моделювання гідроакустичного сигналу для вказаної кількості відштовхувань променів від дна та поверхні водного середовища, що під час розрахунку траєкторії руху променів визначає їх кінцеву довжину після досягнення гідрофону, підраховує фазу по хвилі та на точці гідрофона.

2. Було написано код, що для моделювання гідроакустичного сигналу, визначено фазу хвилі та довжину для наступного знаходження векторів тиску та швидкості у точці знаходження гідрофону. Показники тиску та швидкості у точці розташування гідрофона розраховані за допомогою формул поданих нижче.

$$v(r,t) = \omega\delta_m \cos\cos\left(kr - \omega t - \varphi_{\delta,0} + \frac{\pi}{2}\right) = v_m \cos(kr - \omega t + \varphi_{v,0})$$

$$p(r,t) = pc^2 k\delta_m \cos\cos\left(kr - \omega t - \varphi_{\delta,0} + \frac{\pi}{2}\right) = p_m \cos(kr - \omega t + \varphi_{p,0})$$

3. Збудовано інтерфейс, що сприяє зручній взаємодії користувача з програмним продуктом, що дає можливість вводити початкові вхідні дані для моделювання, знайти глибину гідрофона та весь час моделювання. Також з допомогою цього інтерфейсу користувач може ввести всі дані, які стосуються морських об’єктів, сигнал яких і буде згенерований, а точніше його швидкість, глибина занурення, частота коливань променів та їх амплітуда. Для ліпшого сприйняття даних, отриманих після моделювання, були створені графіки, що відображають найважливішу інформацію у візуалізованому вигляді.

4. Візуалізоване зображення характеристик сигналу, таких як: p, vx, vy та vz. Була реалізована можливість запису вихідних даних до файла в бінарному вигляді. Дні, що ми отримали масштабуються в діапазон цілих чисел -16000-16000 та записується до файла dat розширення в послідовності p, vz, vx, vy.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gulin O.E. and Yaroshchuk I.O. Simulation of underwater acoustical field fluctuations in range-dependent random environment of shallow sea // J. Comp. Acoust. 2014. Vol. 22. No.1, 1440006.

2. Бреховских Л.М., Лысанов Ю.П. Теоретические основы акустики океана. Наука, 2007. - 370 с.

3. В.С. Ковальский, А.Х. Дегтерев. Лучевая имитационная модель распространения звука в морской среде. НАН Украины. МГИ: – Севастополь. 2004. – 320 с.

4. Фримен Адам Angular для профессионалов. - М. СПб: Издательский дом "Питер", 2017. - 800 с.

5. Итан Браун Веб-разработка с применением Node и Express. - М. СПб: Издательский дом "Питер", 2016. - 336 с.

6. Документація Angular [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/

7. Документація Lucy Software [Електронний ресурс] – Режим доступу до ресурсу: http://oceansonics.com/lucy-software/

8. Eric Elliott Programming JavaScript Applications. - 1 edition изд. O'Reilly Media, 2014. - 254 с.

9. Документація HtmlBook [Електронний ресурс] – Режим доступу до ресурсу: http://htmlbook.ru/

10. Руководство Canvas [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/API/Canvas_API/Tutorial

11. Документація по RxJs [Електронний ресурс] – Режим доступу до ресурсу: https://www.learnrxjs.io/

12. Яков Файн, Моисеев Антон Angular и TypeScript. Сайтостроение для профессионалов. Питер, 2017. - 464 с.

13. Николас Закас ECMASCRIPT 6 ДЛЯ РАЗРАБОТЧИКОВ. Питер, 2017. - 352 с.

14. Basarat Ali Syed Beginning Node.js. Apress, 2014. - 308 с.

15. Документація по Node.Js [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/uk/docs/

16. Asynchronous Validation With Angular Reactive Forms [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@kahlil/asynchronous-validation-with-angular-reactive-forms-1a392971c062

ДОДАТОК 1

Web-сервіс генерації гідроакустичного сигналу променевим методом

Специфікація

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ52

Аркушів 2

Київ – 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТМ52 19Б 81-1	Єлісеєв_М. В_ТМ52.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТМ52 19Б 12-1	Server.js	Модулі серверної частини
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТМ52 19Б 12-2	app.component.ts	Модуль клієнтської частини
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТМ52 19Б 12-3	ModellingModule.js	Модуль моделювання сигналу
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТМ52 19Б 12-4	PVCalculatorModule.js	Модуль розрахунку векторної швидкості та тиску
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТМ52 19Б 12-5	RaysModule.js	Модуль отримання променів
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТМ52 19Б 12-6	FileSystemModule.js	Модуль для завантаження сигналу
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТМ52 19Б 13-2	Опис.docx	Опис модуля інтерфейсу клієнтської частини програми

ДОДАТОК 2

Web-сервіс генерації гідроакустичного сигналу променевим методом

Текст програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ52_19Б 12-1

Аркушів 10

Київ – 2019

```
// Код на запит з клієнтської сторони на моделювання
app.post('/api/modeling', (req, res) => {
   res.send({result: modelling.runModelling(req.body.params)});
});

// Код на запит з клієнтської сторони на завантаження
app.get('/api/download', (req, res) => {
   res.download('signal.dat');
});
```



```
function runModelling(payload) {
   // Трансформація направлень в необхідний формат
   payload.arrows = transformObjects(payload.arrows);
   // Визначення кроку
   const timeStep = 1 / payload.frequency;

   // Ініціалізація модулю PVCalculator початковими даними
   pvCalculator.init({ ...payload, reflections: 6 });

   const list = [];
   for (let currentTime = payload.timeStart; currentTime < payload.timeEnd; currentTime
+= timeStep) {
      // Отримання даних про швидкість та тиск в кожен момент часу
      const pv = pvCalculator.getPV(currentTime);
      // Додавання їх у фінальний список в необхідному форматі
      list.push(getTimeSeriesGeneratorElement(currentTime, pv.pressure, pv.velocity));
   }
   // Збереження списку
   return save(list);
}

function save(list) {
   const MAX_VAL = 16000;
   const MIN_VAL = -16000;
   let minV = Number.MAX_VALUE;
   let maxV = -Number.MAX_VALUE;
   let minP = Number.MAX_VALUE;
   let maxP = -Number.MAX_VALUE;
   // Scale
```

```
    list.forEach(el => {
        if (el.pressure > maxP)
            maxP = el.pressure;
        if (el.pressure < minP)
            minP = el.pressure;
        maxV = (el.vx > maxV) ? el.vx : maxV;
        maxV = (el.vy > maxV) ? el.vy : maxV;
        maxV = (el.vz > maxV) ? el.vz : maxV;
        minV = (el.vx < minV) ? el.vx : minV;
        minV = (el.vy < minV) ? el.vy : minV;
        minV = (el.vz < minV) ? el.vz : minV;
    });
    const res = [];

    list.forEach(el => {
        // Отримуємо тиск
        const p = Math.round((el.pressure - minP) / (maxP - minP) * (MAX_VAL -
MIN_VAL) + MIN_VAL);
        // Отримуємо векторну швидкість по осі X
        const vx = Math.round((el.vx - minV) / (maxV - minV) * (MAX_VAL - MIN_VAL)
+ MIN_VAL);
        // Отримуємо векторну швидкість по осі Y
        const vy = Math.round((el.vy - minV) / (maxV - minV) * (MAX_VAL - MIN_VAL)
+ MIN_VAL);
        // Отримуємо векторну швидкість по осі Z
        const vz = Math.round((el.vz - minV) / (maxV - minV) * (MAX_VAL - MIN_VAL)
+ MIN_VAL);
        res.push({ p, vx, vy, vz, time: el.currentTime });
    });
    // Зберігаємо у файл через модуль файлової системи
    return fsModule.save(res);
}

function getTimeSeriesGeneratorElement(currentTime, pressure, velocity) {
    return {
        currentTime,
        pressure: pressure.z,
        vx: velocity.x,
        vy: velocity.y,
        vz: velocity.z
    }
}

function transformObjects(arrows) {
    const frequencies = [{ frequency: 40, amplitude: 1e-5 }];
```

```
    return arrows.map(arrow => ({ ...arrow, frequencies }))
}
```



```
function getPV(currentTime) {
    let res = {};

    objects.forEach(obj => {
        // Знаходження всіх променів відбиття в даний час
        const rays = rayTracer.getRays(getPosition(obj, currentTime), gasPosition,
seaDepth);
        obj.frequencies.forEach(freq => {
            rays.forEach(ray =>{
                // Дистанція від променя до вектора
                const r = getDistanceVector(ray);
                const dist = vectorModule.getLength(r);
                // Амплітуда
                const a = getAmplitude(freq.amplitude, dist);
                // Довжина хвилі
                const waveLen = getWaveLength(freq.frequency);
                // Вектор хвилі
                const k = getWaveVector(waveLen, r);
                // Зміна фази
                const phaseChange = getPhaseChange(freq.frequency, dist);
                // Швидкість
                const v = getVelocity(freq.frequency, a, k, r, currentTime, phaseChange);
                // Тиск
                const p = getPressure(freq.frequency, a, k, r, currentTime, phaseChange);
                res.velocity = res.velocity ? vectorModule.add(res.velocity, v) : v;
                res.pressure = res.pressure ? vectorModule.add(res.pressure, p) : p;
            });
        });
    });
    return res;
}

function getPosition(obj, time) {
    const beginLoc = vectorModule.getVector(obj.beginLocation.layerX,
obj.beginLocation.layerY, obj.beginLocation.depth);
    const targetLoc= vectorModule.getVector(obj.targetLocation.layerX,
obj.targetLocation.layerY, obj.targetLocation.depth);
    return vectorModule.add(beginLoc,
vectorModule.multiple(vectorModule.multiple(vectorModule.unitary(vectorModule.subtra
```

```
ct(targetLoc, beginLoc)), +obj.velocity), +time));
}

function getAmplitude(a0, dist) {
   return dist < Number.EPSILON
      ? a0
      : a0 * Math.exp(-amplitude * dist);
}

function getDistanceVector(ray) {
   const rayLen = ray.length;
   const n = ray.points.length;
   const dir = vectorModule.unitary(vectorModule.subtract(ray.points[n - 1], ray.points[n -
2]));
   return vectorModule.multiple(dir, rayLen);
}

function getWaveLength(waveFrequency) {
   return soundSpeed * waveFrequency;
}

function getWaveVector(waveLen, r) {
   const k = getWaveNumber(waveLen);
   return vectorModule.multiple(vectorModule.unitary(r), k);
}

function getWaveNumber(waveLen) {
   return 2 * Math.PI / waveLen;
}

function getPhaseChange(waveFrequency, distance) {
   return 2 * Math.PI * distance * waveFrequency / soundSpeed;
}

function getVelocity(frequency, a, k, r, t, phaseShift) {
   const w = frequency * 2 * Math.PI;
   const amp = w * a;
   const phi = vectorModule.multiple(k, r) - w * t + phaseShift + Math.PI;
   return getRotatedVectorWithLength(amp, phi, r);
}

function getPressure(frequency, a, k, r, t, phaseShift) {
   const w = frequency * 2 * Math.PI;
   const amp = r0 * soundSpeed * soundSpeed * vectorModule.getLength(k) * a;
   const phi = vectorModule.multiple(k, r) - w * t + phaseShift + Math.PI / 2;
```

```
    return getRotatedVectorWithLength(amp, phi, r);
}

function getRotatedVectorWithLength(max, phase, R)
{
    const xy = vectorModule.getVector(R.x, R.y, 0);
    const planeVector = vectorModule.getVector(vectorModule.getLength(xy), R.z, 0);
    const rotPlaneVector = vectorModule.rotateByZ(planeVector, phase);
    const resVector = vectorModule.multiple(vectorModule.unitary(xy), rotPlaneVector.x);
    resVector.z = rotPlaneVector.y;
    return vectorModule.multiple(vectorModule.unitary(resVector), max);
}
```



```
function getRays(objPos, gasPos, seaDepth) {
    if (!objPos)
        throw Error("objectPos must be not null");
    if (!gasPos)
        throw Error("gasPos must be not null");
    const res = [];
    const dirToObj = vectorModule.subtract(objPos, gasPos);
    const dirToObjXY = vectorModule.getVector(dirToObj.x, dirToObj.y);
    const relObj = vectorModule.getVector(vectorModule.getLength(dirToObjXY),
objPos.z);
    const relGas = vectorModule.getVector(0, gasPos.z);

    for (let reflections = -gReflections; reflections <= gReflections; reflections++)
    {
        // Find Relative Points
        const ray = getRelativeRayPoints(seaDepth, relObj, relGas, reflections);
        if (ray == null)
            continue;

        // Add other
        const toTopRealPoints = [];
        ray.forEach(relPoint => {
            const objWithoutDepth = vectorModule.add(gasPos,
vectorModule.multiple(vectorModule.unitary(dirToObjXY), relPoint.x));
            toTopRealPoints.push(vectorModule.getVector(objWithoutDepth.x,
objWithoutDepth.y, relPoint.y));
        });
        const toTopReal = trajectoryModule.getTrajectory(toTopRealPoints);
```

```
        res.push(toTopReal);
    }
    return res;
}

function getRelativeRayPoints(bottomDepth, start, end, reflections = 0) {
    if (!reflections) {
        return [ { ...start }, { ...end } ];
    }
    const isRayToTop = reflections <  0;
    reflections = Math.abs(reflections);
    if ((Math.abs(start.y) < Number.EPSILON && isRayToTop)
        || (Math.abs(bottomDepth - start.y) < Number.EPSILON && !isRayToTop)
        || (Math.abs(end.x - start.x) < Number.EPSILON && !reflections))
    {
        return null;
    }

    const res = [ { ...start } ];

    let bottomIsCurrentReflector = ((reflections % 2 === 1) && !isRayToTop) ||
(!(reflections % 2 === 1) && isRayToTop) ;

    const imageTo = { ...end };
    const images = [];
    for (let i = 0; i < reflections; i++) {
        // make reflection
        imageTo.y = bottomIsCurrentReflector
            ? bottomDepth + bottomDepth - imageTo.y
            : -imageTo.y;

        images.push({ ...imageTo });
        bottomIsCurrentReflector = !bottomIsCurrentReflector;
    }

    bottomIsCurrentReflector = !bottomIsCurrentReflector;

    let currentFrom = { ...start };
    for (let i = images.length - 1; i >= 0; i--)
    {
        // Current image
        let currentImageTo = images[i];
        // Find reflection point
        if (bottomIsCurrentReflector)
```

```
        {
            currentImageTo.x = currentFrom.x - currentFrom.x * (bottomDepth -
currentFrom.y) / (currentImageTo.y - currentFrom.y);
            currentImageTo.y = bottomDepth;
        }
        else
        {
            currentImageTo.x = currentFrom.x - currentFrom.x * currentFrom.y /
(currentFrom.y - currentImageTo.y);
            currentImageTo.y = 0;
        }


        currentFrom = currentImageTo;
        bottomIsCurrentReflector = !bottomIsCurrentReflector;
        res.push({ ...currentImageTo });
    }
    res.push({ ...end });

    return res;
}
```



```
// Збереження сигналу в файл
function save(res) {
    const response = writeBinary(res);
    fs.writeFile('./signal.dat', response, e => {
        console.log(e || 'File successfully loaded.');
    });
    return res;
}

// Бінарний запис в файл
function writeBinary(res) {
    res.forEach(el => {
        bw.writeInt8(el.p);
        bw.writeInt8(el.vx);
        bw.writeInt8(el.vy);
        bw.writeInt8(el.vz);
    });
    return bw.toBuffer();
}
```

```
modeling(signalFormData: SignalFormInterface): void {
  const req = {
    ...signalFormData,
    arrows: this.arrows,
    pointWithoutDirection: this.pointWithoutDirection,
  };
  this.inProgress$.next(true);
  this.apiService.modeling(req).subscribe(
      res => {
          console.log(res);
          this.inProgress$.next(false);
          const shouldDownload = confirm('Файл успешно сгенерирован! Скачать файл?');
          if (shouldDownload) {
              const link = document.getElementById('download-link');
              link.click();
          }
          this.dataToBuildCharts = res;
      },
      e => alert(e.message || e),
  );
}
```

```
modeling(signalFormData: SignalFormInterface): void {
  // Запис в змінну для запиту з правильною структурою
  const req = {
    ...signalFormData,
    arrows: this.arrows,
    pointWithoutDirection: this.pointWithoutDirection,
  };
  this.inProgress$.next(true);
  // Звернення до API сервісу для запиту на початок моделювання
  this.apiService.modeling(req).subscribe(
      // Успішний сценарій моделювання
      res => {
          this.inProgress$.next(false);
          const shouldDownload = confirm('Файл успешно сгенерирован! Скачать файл?');
          if (shouldDownload) {
              // Початок завантаження файлу з сервера
              const link = document.getElementById('download-link');
              link.click();
          }
```

```
        // Збереження даних про сигнал
        this.dataToBuildCharts = res;
      },
      // Обробка помилки
      e => alert(e.message || e),
  );
}

// Метод для посилання запиту на серверну частину для початку моделювання
modeling(payload: ModelingInterface): Observable<ModellingResponseDataInterface[]>
{
  return
this.http.post<ModellingResponseDataInterface[]>('http://localhost:8000/api/modeling', {
params: payload, observe: 'response' }).pipe(
      pluck('result'),
  );
}


// Створення форми для введення початкових даних з заданими дефолтними даними
private createForm(): void {
  this.signalForm = this.fb.group({
    timeStart: [
      0,
      Validators.required,
    ],
    timeEnd: [
      1,
      Validators.required,
    ],
    gasDepth: [
      450,
      Validators.required,

    ],
    seaDepth: [
      500,
      Validators.required,

    ],
    amplitude: [
      0.002,
      Validators.required,
    ],
    frequency: [
```

```
      1024,
      Validators.required,
    ],
    soundSpeed: [
      1500,
      Validators.required,
    ],
  });
}
```

```
< <!--Загльна структура сервісу-->

<!--Лоадер-->
<mat-progress-bar class="progress-bar" mode="indeterminate"
*ngIf="dataService.inProgress$ | async"></mat-progress-bar>
<!--Контент-->
<div class="global-content">
    <div class="d-flex">
        <div class="flex-1">
            <!--Компонент з головною формою-->
            <app-input-form></app-input-form>
        </div>
        <div class="flex-1">
            <!--Компонент з акваторією-->
            <app-input-chart></app-input-chart>
        </div>
    </div>
    <!--Компонент для виводу таблиці з направленнями-->
    <app-input-appearence></app-input-appearence>
</div>

<!--Невидима силка для скачування згенерованого-->
<a style="display: none" id="download-link" href="http://localhost:8000/api/download"
download></a>
```

ДОДАТОК 3

Web-сервіс генерації гідроакустичного сигналу променевим методом

Опис програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ52_19Б 13-2

Аркушів 8

Київ – 2019

АНОТАЦІЯ

Додаток надає можливість моделювати та генерувати гідроакустичні сигнали променевим методом.

Розроблене програмне забезпечення дозволяє отримати візуальне представлення сигналу у вигляді графіку та звукове представлення у вигляді файлу dat розширення після введення початкових даних.

Web-сервіс було розроблено за допомогою платформи JetBrains PhpStorm 2018.3.3 з використанням мови Node.js для серверної сторони програми та мови Javascript з фреймворком Angular 7 для клієнтської.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Відповідно до теми дипломної роботи, програма має назву «Сервіс генерації гідроакусничного сигналу».

Програма працює через браузер та потребує доступу до мережі інтернет, але не потребує встановлення на ПК зайвого ПО, що забезпечує практичність та швидкість.

Система була написана мовою Javascript з використанням фреймворку Angular 7 та Node.js.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений програмний засіб покликаний вирішити задачу виявлення рельєфу морського дна, навігації під водою тощо. Це було реалізовано за допомогою кількох функцій системи, а саме:

– моделювання гідроакустичного сигналу за допомогою розробленого алгоритму;

– завантаження згенерованого файлу;

– відображення графічної візуалізації гідроакустичного сигналу

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Програмний продукт складається з клієнтської та серверної частини.

Загальний принцип роботи додатку такий:

1) користувач вводить початкові дані;
2) Натискає на кнопку “Моделировать”
3) метод, що викликається в обробнику виконує запит;
4) результати виконання методу виводяться на сторінку.

В першу чергу завантажується форма з усіма її компонентами: кнопки, текстові поля, таблиці та акваторія. Користувач вводить всі необхідні дані та додає направлення на акваторії.

Після натискання на кнопку “Моделировать”, що розміщена під формою, викликається метод modelling, в який передаються всі початкові параметри. Він виконує запит до сервера та передає всі ці дані. Сервер в свою чергу починає проведення розрахунків за написаним алгоритмом. Після успішного завершення він повертає результат на клієнтську сторону у вигляді масиву з порахованими показниками сигналу для його візуалізації та силку на скачування згенерованого файлу з dat розширенням.

Після цього користувач може завантажити даний сигнал через діалогове вікно завантаження браузера, яке відкриється зразу після генерації, або подивитись візуалізацію сигнала, натиснувши на конпку “Посмотреть графики”.

Після закінчення роботи з цими даними, користувач без перезавантаження сторінки може очистити всі введені початкові дані і направлення відповідними кнопками та почати генерацію наступного сигналу.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для організації доступу до програмного продукту потрібно мати комп’ютер або ноутбук.

Кінцевим користувачам для роботи з програмою потрібно, щоб на комп’ютері був встановлений браузер, Node.js та Angular CLI.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними є:

— глибина водного середовища;

— початковий час;

— кінцевий час;

— глибина гідрофона;

— частота коливань;

— амплітуда коливань;

— направлення та відомості про них.

Вихідними даними є:

— згенерований гідроакустичний сигнал у dat файл;

— візуалізація сигналу у вигляді графіка.